InfluxDB Documentation

Release 5.3.1

John Shahid

Nov 14, 2022

Contents

Release 5.3.1

Date Nov 14, 2022

Keywords python, time series, database

CHAPTER 1

Contents

1.1 InfluxDB-Python

1.1.1 InfluxDB-Python

InfluxDB-Python is a client for interacting with InfluxDB.

Development of this library is maintained by:

Github ID	URL
@aviau	(https://github.com/aviau)
@xginn8	(https://github.com/xginn8)
@sebito91	(https://github.com/sebito91)

InfluxDB is an open-source distributed time series database, find more about InfluxDB at https://docs.influxdata.com/influxdb/latest

InfluxDB pre v1.1.0 users

This module is tested with InfluxDB versions: v1.2.4, v1.3.9, v1.4.3, v1.5.4, v1.6.4, and 1.7.4.

Those users still on InfluxDB v0.8.x users may still use the legacy client by importing `from influxdb.influxdb08 import InfluxDBClient`.

Installation

Install, upgrade and uninstall influxdb-python with these commands:

```
$ pip install influxdb
$ pip install --upgrade influxdb
$ pip uninstall influxdb
```

On Debian/Ubuntu, you can install it with this command:

```
$ sudo apt-get install python-influxdb
```

Dependencies

The influxdb-python distribution is supported and tested on Python 2.7, 3.5, 3.6, 3.7, PyPy and PyPy3.

Note: Python <3.5 are currently untested. See `.travis.yml`.

Main dependency is:

- Requests: HTTP library for human beings (http://docs.python-requests.org/)

Additional dependencies are:

- pandas: for writing from and reading to DataFrames (http://pandas.pydata.org/)
- Sphinx: Tool to create and manage the documentation (http://sphinx-doc.org/)
- Nose: to auto-discover tests (http://nose.readthedocs.org/en/latest/)
- Mock: to mock tests (https://pypi.python.org/pypi/mock)

Documentation

Documentation is available at https://influxdb-python.readthedocs.io/en/latest/.

You will need Sphinx installed to generate the documentation.

The documentation can be generated by running:

```
$ tox -e docs
```

Generated documentation can be found in the *docs/build/html/* directory.

Examples

Here's a basic example (for more see the examples directory):

```
$ python

>>> from influxdb import InfluxDBClient

>>> json_body = [
    {
        "measurement": "cpu_load_short",
        "tags": {
            "host": "server01",
            "region": "us-west"
        },
        "time": "2009-11-10T23:00:00Z",
        "fields": {
            "value": 0.64
        }
    }
]

>>> client = InfluxDBClient('localhost', 8086, 'root', 'root', 'example')
```

```
>>> client.create_database('example')

>>> client.write_points(json_body)

>>> result = client.query('select value from cpu_load_short;')

>>> print("Result: {0}".format(result))
```

Testing

Make sure you have tox by running the following:

```
$ pip install tox
```

To test influxdb-python with multiple version of Python, you can use Tox:

```
$ tox
```

Support

For issues with, questions about, or feedback for InfluxDB, please look into our community page: http://influxdb.com/community/.

We are also lurking on the following:

- #influxdb on irc.freenode.net
- #influxdb on gophers.slack.com

Development

All development is done on Github. Use Issues to report problems or submit contributions.

Please note that we WILL get to your questions/issues/concerns as quickly as possible. We maintain many software repositories and sometimes things may get pushed to the backburner. Please don't take offense, we will do our best to reply as soon as possible!

Source code

The source code is currently available on Github: https://github.com/influxdata/influxdb-python

TODO

The TODO/Roadmap can be found in Github bug tracker: https://github.com/influxdata/influxdb-python/issues

1.2 API Documentation

To connect to a InfluxDB, you must create a *InfluxDBClient* object. The default configuration connects to InfluxDB on `localhost` with the default ports. The below instantiation statements are all equivalent:

```
from influxdb import InfluxDBClient

# using Http
client = InfluxDBClient(database='dbname')
client = InfluxDBClient(host='127.0.0.1', port=8086, database='dbname')
client = InfluxDBClient(host='127.0.0.1', port=8086, username='root', password='root',
↪ database='dbname')

# using UDP
client = InfluxDBClient(host='127.0.0.1', database='dbname', use_udp=True, udp_
↪port=4444)
```

To write pandas DataFrames or to read data into a pandas DataFrame, use a *DataFrameClient* object. These clients are initiated in the same way as the *InfluxDBClient*:

```
from influxdb import DataFrameClient

client = DataFrameClient(host='127.0.0.1', port=8086, username='root', password='root
↪', database='dbname')
```

Note: Only when using UDP (use_udp=True) the connection is established.

1.2.1 InfluxDBClient

class influxdb.**InfluxDBClient**(*host=u'localhost'*, *port=8086*, *username=u'root'*, *password=u'root'*, *database=None*, *ssl=False*, *verify_ssl=False*, *timeout=None*, *retries=3*, *use_udp=False*, *udp_port=4444*, *proxies=None*, *pool_size=10*, *path=u''*, *cert=None*, *gzip=False*, *session=None*, *headers=None*)

InfluxDBClient primary client object to connect InfluxDB.

The *InfluxDBClient* object holds information necessary to connect to InfluxDB. Requests can be made to InfluxDB directly through the client.

The client supports the use as a context manager.

Parameters

- **host** (*str*) – hostname to connect to InfluxDB, defaults to 'localhost'
- **port** (*int*) – port to connect to InfluxDB, defaults to 8086
- **username** (*str*) – user to connect, defaults to 'root'
- **password** (*str*) – password of the user, defaults to 'root'
- **pool_size** (*int*) – urllib3 connection pool size, defaults to 10.
- **database** (*str*) – database name to connect to, defaults to None
- **ssl** (*bool*) – use https instead of http to connect to InfluxDB, defaults to False
- **verify_ssl** (*bool*) – verify SSL certificates for HTTPS requests, defaults to False
- **timeout** (*int*) – number of seconds Requests will wait for your client to establish a connection, defaults to None

- **retries** (*int*) – number of attempts your client will make before aborting, defaults to 3 0 - try until success 1 - attempt only once (without retry) 2 - maximum two attempts (including one retry) 3 - maximum three attempts (default option)
- **use_udp** (*bool*) – use UDP to connect to InfluxDB, defaults to False
- **udp_port** (*int*) – UDP port to connect to InfluxDB, defaults to 4444
- **proxies** (*dict*) – HTTP(S) proxy to use for Requests, defaults to {}
- **path** (*str*) – path of InfluxDB on the server to connect, defaults to ''
- **cert** (*str*) – Path to client certificate information to use for mutual TLS authentication. You can specify a local cert to use as a single file containing the private key and the certificate, or as a tuple of both files' paths, defaults to None
- **gzip** (*bool*) – use gzip content encoding to compress requests
- **session** (*requests.Session*) – allow for the new client request to use an existing requests Session, defaults to None
- **headers** (*dict*) – headers to add to Requests, will add 'Content-Type' and 'Accept' unless these are already present, defaults to {}

Raises **ValueError** – if cert is provided but ssl is disabled (set to False)

alter_retention_policy(*name*, *database=None*, *duration=None*, *replication=None*, *default=None*, *shard_duration=None*)

Modify an existing retention policy for a database.

Parameters

- **name** (*str*) – the name of the retention policy to modify
- **database** (*str*) – the database for which the retention policy is modified. Defaults to current client's database
- **duration** (*str*) – the new duration of the existing retention policy. Durations such as 1h, 90m, 12h, 7d, and 4w, are all supported and mean 1 hour, 90 minutes, 12 hours, 7 day, and 4 weeks, respectively. For infinite retention, meaning the data will never be deleted, use 'INF' for duration. The minimum retention period is 1 hour.
- **replication** (*int*) – the new replication of the existing retention policy
- **default** (*bool*) – whether or not to set the modified policy as default
- **shard_duration** (*str*) – the shard duration of the retention policy. Durations such as 1h, 90m, 12h, 7d, and 4w, are all supported and mean 1 hour, 90 minutes, 12 hours, 7 day, and 4 weeks, respectively. Infinite retention is not supported. As a workaround, specify a "1000w" duration to achieve an extremely long shard group duration. The minimum shard group duration is 1 hour.

Note: at least one of duration, replication, or default flag should be set. Otherwise the operation will fail.

---

close()

Close http session.

create_continuous_query(*name*, *select*, *database=None*, *resample_opts=None*)

Create a continuous query for a database.

Parameters

- **name** (*str*) – the name of continuous query to create

- **select** (*str*) – select statement for the continuous query
- **database** (*str*) – the database for which the continuous query is created. Defaults to current client's database
- **resample_opts** (*str*) – resample options

Example

```
>> select_clause = 'SELECT mean("value") INTO "cpu_mean" ' \
...                 'FROM "cpu" GROUP BY time(1m)'
>> client.create_continuous_query(
...     'cpu_mean', select_clause, 'db_name', 'EVERY 10s FOR 2m'
... )
>> client.get_list_continuous_queries()
[
    {
        'db_name': [
            {
                'name': 'cpu_mean',
                'query': 'CREATE CONTINUOUS QUERY "cpu_mean" '
                         'ON "db_name" '
                         'RESAMPLE EVERY 10s FOR 2m '
                         'BEGIN SELECT mean("value") '
                         'INTO "cpu_mean" FROM "cpu" '
                         'GROUP BY time(1m) END'
            }
        ]
    }
]
```

create_database(*dbname*)

Create a new database in InfluxDB.

Parameters **dbname** (*str*) – the name of the database to create

create_retention_policy(*name*, *duration*, *replication*, *database=None*, *default=False*, *shard_duration=u'0s'*)

Create a retention policy for a database.

Parameters

- **name** (*str*) – the name of the new retention policy
- **duration** (*str*) – the duration of the new retention policy. Durations such as 1h, 90m, 12h, 7d, and 4w, are all supported and mean 1 hour, 90 minutes, 12 hours, 7 day, and 4 weeks, respectively. For infinite retention - meaning the data will never be deleted - use 'INF' for duration. The minimum retention period is 1 hour.
- **replication** (*str*) – the replication of the retention policy
- **database** (*str*) – the database for which the retention policy is created. Defaults to current client's database
- **default** (*bool*) – whether or not to set the policy as default
- **shard_duration** (*str*) – the shard duration of the retention policy. Durations such as 1h, 90m, 12h, 7d, and 4w, are all supported and mean 1 hour, 90 minutes, 12 hours, 7 day, and 4 weeks, respectively. Infinite retention is not supported. As a workaround, specify a "1000w" duration to achieve an extremely long shard group duration. Defaults to "0s", which is interpreted by the database to mean the default value given the duration. The minimum shard group duration is 1 hour.

create_user(*username*, *password*, *admin=False*)
Create a new user in InfluxDB.

Parameters

- **username** (*str*) – the new username to create
- **password** (*str*) – the password for the new user
- **admin** (*boolean*) – whether the user should have cluster administration privileges or not

delete_series(*database=None*, *measurement=None*, *tags=None*)
Delete series from a database.

Series must be filtered by either measurement and tags. This method cannot be used to delete all series, use *drop_database* instead.

Parameters

- **database** (*str*) – the database from which the series should be deleted, defaults to client's current database
- **measurement** (*str*) – Delete all series from a measurement
- **tags** (*dict*) – Delete all series that match given tags

drop_continuous_query(*name*, *database=None*)
Drop an existing continuous query for a database.

Parameters

- **name** (*str*) – the name of continuous query to drop
- **database** (*str*) – the database for which the continuous query is dropped. Defaults to current client's database

drop_database(*dbname*)
Drop a database from InfluxDB.

Parameters **dbname** (*str*) – the name of the database to drop

drop_measurement(*measurement*)
Drop a measurement from InfluxDB.

Parameters **measurement** (*str*) – the name of the measurement to drop

drop_retention_policy(*name*, *database=None*)
Drop an existing retention policy for a database.

Parameters

- **name** (*str*) – the name of the retention policy to drop
- **database** (*str*) – the database for which the retention policy is dropped. Defaults to current client's database

drop_user(*username*)
Drop a user from InfluxDB.

Parameters **username** (*str*) – the username to drop

classmethod from_dsn(*dsn*, ***kwargs*)
Generate an instance of InfluxDBClient from given data source name.

Return an instance of *InfluxDBClient* from the provided data source name. Supported schemes are "influxdb", "https+influxdb" and "udp+influxdb". Parameters for the *InfluxDBClient* constructor may also be passed to this method.

Parameters

- **dsn** (*string*) – data source name
- **kwargs** (*dict*) – additional parameters for *InfluxDBClient*

Raises **ValueError** – if the provided DSN has any unexpected values

Example

```
>> cli = InfluxDBClient.from_dsn('influxdb://username:password@\
localhost:8086/databasename', timeout=5)
>> type(cli)
<class 'influxdb.client.InfluxDBClient'>
>> cli = InfluxDBClient.from_dsn('udp+influxdb://username:pass@\
localhost:8086/databasename', timeout=5, udp_port=159)
>> print('{0._baseurl} - {0.use_udp} {0.udp_port}'.format(cli))
http://localhost:8086 - True 159
```

Note: parameters provided in ***kwargs* may override dsn parameters

Note: when using "udp+influxdb" the specified port (if any) will be used for the TCP connection; specify the UDP port with the additional *udp_port* parameter (cf. examples).

get_list_continuous_queries()

Get the list of continuous queries in InfluxDB.

Returns all CQs in InfluxDB

Return type list of dictionaries

Example

```
>> cqs = client.get_list_cqs()
>> cqs
[
    {
        u'db1': []
    },
    {
        u'db2': [
            {
                u'name': u'vampire',
                u'query': u'CREATE CONTINUOUS QUERY vampire ON '
                          'mydb BEGIN SELECT count(dracula) INTO '
                          'mydb.autogen.all_of_them FROM '
                          'mydb.autogen.one GROUP BY time(5m) END'
            }
        ]
    }
]
```

get_list_database()

Get the list of databases in InfluxDB.

Returns all databases in InfluxDB

Return type list of dictionaries

Example

```
>> dbs = client.get_list_database()
>> dbs
[{u'name': u'db1'}, {u'name': u'db2'}, {u'name': u'db3'}]
```

get_list_measurements()
Get the list of measurements in InfluxDB.

Returns all measurements in InfluxDB

Return type list of dictionaries

Example

```
>> dbs = client.get_list_measurements()
>> dbs
[{u'name': u'measurements1'},
 {u'name': u'measurements2'},
 {u'name': u'measurements3'}]
```

get_list_privileges(*username*)
Get the list of all privileges granted to given user.

Parameters **username** (*str*) – the username to get privileges of

Returns all privileges granted to given user

Return type list of dictionaries

Example

```
>> privileges = client.get_list_privileges('user1')
>> privileges
[{u'privilege': u'WRITE', u'database': u'db1'},
 {u'privilege': u'ALL PRIVILEGES', u'database': u'db2'},
 {u'privilege': u'NO PRIVILEGES', u'database': u'db3'}]
```

get_list_retention_policies(*database=None*)
Get the list of retention policies for a database.

Parameters **database** (*str*) – the name of the database, defaults to the client's current database

Returns all retention policies for the database

Return type list of dictionaries

Example

```
>> ret_policies = client.get_list_retention_policies('my_db')
>> ret_policies
[{u'default': True,
  u'duration': u'0',
  u'name': u'default',
  u'replicaN': 1}]
```

get_list_series(*database=None*, *measurement=None*, *tags=None*)
Query SHOW SERIES returns the distinct series in your database.

FROM and WHERE clauses are optional.

Parameters

- **measurement** – Show all series from a measurement
- **tags** – Show all series that match given tags
- **database** (*str*) – the database from which the series should be shows, defaults to client's current database

get_list_users()
Get the list of all users in InfluxDB.

Returns all users in InfluxDB

Return type list of dictionaries

Example

```
>> users = client.get_list_users()
>> users
[{u'admin': True, u'user': u'user1'},
 {u'admin': False, u'user': u'user2'},
 {u'admin': False, u'user': u'user3'}]
```

grant_admin_privileges(*username*)
Grant cluster administration privileges to a user.

Parameters **username** (*str*) – the username to grant privileges to

Note: Only a cluster administrator can create/drop databases and manage users.

grant_privilege(*privilege*, *database*, *username*)
Grant a privilege on a database to a user.

Parameters

- **privilege** (*str*) – the privilege to grant, one of 'read', 'write' or 'all'. The string is case-insensitive
- **database** (*str*) – the database to grant the privilege on
- **username** (*str*) – the username to grant the privilege to

ping()
Check connectivity to InfluxDB.

Returns The version of the InfluxDB the client is connected to

query(*query*, *params=None*, *bind_params=None*, *epoch=None*, *expected_response_code=200*, *database=None*, *raise_errors=True*, *chunked=False*, *chunk_size=0*, *method=u'GET'*)
Send a query to InfluxDB.

Danger: In order to avoid injection vulnerabilities (similar to SQL injection vulnerabilities), do not directly include untrusted data into the `query` parameter, use `bind_params` instead.

Parameters

- **query** (*str*) – the actual query string

- **params** (*dict*) – additional parameters for the request, defaults to {}
- **bind_params** (*dict*) – bind parameters for the query: any variable in the query written as `'$var_name'` will be replaced with `bind_params['var_name']`. Only works in the `WHERE` clause and takes precedence over `params['params']`
- **epoch** (*str*) – response timestamps to be in epoch format either 'h', 'm', 's', 'ms', 'u', or 'ns',defaults to *None* which is RFC3339 UTC format with nanosecond precision
- **expected_response_code** (*int*) – the expected status code of response, defaults to 200
- **database** (*str*) – database to query, defaults to None
- **raise_errors** (*bool*) – Whether or not to raise exceptions when InfluxDB returns errors, defaults to True
- **chunked** (*bool*) – Enable to use chunked responses from InfluxDB. With `chunked` enabled, one ResultSet is returned per chunk containing all results within that chunk
- **chunk_size** (*int*) – Size of each chunk to tell InfluxDB to use.
- **method** (*str*) – the HTTP method for the request, defaults to GET

Returns the queried data

Return type *ResultSet*

request(*url*, *method=u'GET'*, *params=None*, *data=None*, *stream=False*, *expected_response_code=200*, *headers=None*)

Make a HTTP request to the InfluxDB API.

Parameters

- **url** (*str*) – the path of the HTTP request, e.g. write, query, etc.
- **method** (*str*) – the HTTP method for the request, defaults to GET
- **params** (*dict*) – additional parameters for the request, defaults to None
- **data** (*str*) – the data of the request, defaults to None
- **stream** (*bool*) – True if a query uses chunked responses
- **expected_response_code** (*int*) – the expected response code of the request, defaults to 200
- **headers** (*dict*) – headers to add to the request

Returns the response from the request

Return type `requests.Response`

Raises

- ***InfluxDBServerError*** – if the response code is any server error code (5xx)
- ***InfluxDBClientError*** – if the response code is not the same as *expected_response_code* and is not a server error code

revoke_admin_privileges(*username*)

Revoke cluster administration privileges from a user.

Parameters **username** (*str*) – the username to revoke privileges from

Note: Only a cluster administrator can create/ drop databases and manage users.

revoke_privilege(*privilege*, *database*, *username*)
Revoke a privilege on a database from a user.

Parameters

- **privilege** (*str*) – the privilege to revoke, one of 'read', 'write' or 'all'. The string is case-insensitive
- **database** (*str*) – the database to revoke the privilege on
- **username** (*str*) – the username to revoke the privilege from

send_packet(*packet*, *protocol=u'json'*, *time_precision=None*)
Send an UDP packet.

Parameters

- **packet** (*(if protocol is 'json') dict (if protocol is 'line') list of line protocol strings*) – the packet to be sent
- **protocol** (*str*) – protocol of input data, either 'json' or 'line'
- **time_precision** (*str*) – Either 's', 'm', 'ms' or 'u', defaults to None

set_user_password(*username*, *password*)
Change the password of an existing user.

Parameters

- **username** (*str*) – the username who's password is being changed
- **password** (*str*) – the new password for the user

switch_database(*database*)
Change the client's database.

Parameters **database** (*str*) – the name of the database to switch to

switch_user(*username*, *password*)
Change the client's username.

Parameters

- **username** (*str*) – the username to switch to
- **password** (*str*) – the password for the username

write(*data*, *params=None*, *expected_response_code=204*, *protocol=u'json'*)
Write data to InfluxDB.

Parameters

- **data** – the data to be written
- **params** (*dict*) – additional parameters for the request, defaults to None
- **expected_response_code** (*int*) – the expected response code of the write operation, defaults to 204
- **protocol** (*str*) – protocol of input data, either 'json' or 'line'

Returns True, if the write operation is successful

Return type bool

write_points(*points*, *time_precision=None*, *database=None*, *retention_policy=None*, *tags=None*, *batch_size=None*, *protocol=u'json'*, *consistency=None*)

Write to multiple time series names.

Parameters

- **points** (*list of dictionaries, each dictionary represents a point*) – the list of points to be written in the database
- **time_precision** (*str*) – Either 's', 'm', 'ms' or 'u', defaults to None
- **database** (*str*) – the database to write the points to. Defaults to the client's current database
- **tags** (*dict*) – a set of key-value pairs associated with each point. Both keys and values must be strings. These are shared tags and will be merged with point-specific tags, defaults to None
- **retention_policy** (*str*) – the retention policy for the points. Defaults to None
- **batch_size** (*int*) – value to write the points in batches instead of all at one time. Useful for when doing data dumps from one database to another or when doing a massive write operation, defaults to None
- **protocol** (*str*) – Protocol for writing data. Either 'line' or 'json'.
- **consistency** (*str*) – Consistency for the points. One of {'any','one','quorum','all'}.

Returns True, if the operation is successful

Return type bool

Note: if no retention policy is specified, the default retention policy for the database is used

1.2.2 DataFrameClient

class influxdb.**DataFrameClient**(*host=u'localhost'*, *port=8086*, *username=u'root'*, *password=u'root'*, *database=None*, *ssl=False*, *verify_ssl=False*, *timeout=None*, *retries=3*, *use_udp=False*, *udp_port=4444*, *proxies=None*, *pool_size=10*, *path=u''*, *cert=None*, *gzip=False*, *session=None*, *headers=None*)

DataFrameClient instantiates InfluxDBClient to connect to the backend.

The `DataFrameClient` object holds information necessary to connect to InfluxDB. Requests can be made to InfluxDB directly through the client. The client reads and writes from pandas DataFrames.

EPOCH = Timestamp('1970-01-01 00:00:00+0000', tz='UTC')

query(*query*, *params=None*, *bind_params=None*, *epoch=None*, *expected_response_code=200*, *database=None*, *raise_errors=True*, *chunked=False*, *chunk_size=0*, *method=u'GET'*, *dropna=True*, *data_frame_index=None*)

Query data into a DataFrame.

Danger: In order to avoid injection vulnerabilities (similar to SQL injection vulnerabilities), do not directly include untrusted data into the `query` parameter, use `bind_params` instead.

Parameters

- **query** – the actual query string
- **params** – additional parameters for the request, defaults to {}
- **bind_params** – bind parameters for the query: any variable in the query written as `'$var_name'` will be replaced with `bind_params['var_name']`. Only works in the `WHERE` clause and takes precedence over `params['params']`
- **epoch** – response timestamps to be in epoch format either 'h', 'm', 's', 'ms', 'u', or 'ns',defaults to *None* which is RFC3339 UTC format with nanosecond precision
- **expected_response_code** – the expected status code of response, defaults to 200
- **database** – database to query, defaults to None
- **raise_errors** – Whether or not to raise exceptions when InfluxDB returns errors, defaults to True
- **chunked** – Enable to use chunked responses from InfluxDB. With `chunked` enabled, one ResultSet is returned per chunk containing all results within that chunk
- **chunk_size** – Size of each chunk to tell InfluxDB to use.
- **dropna** – drop columns where all values are missing
- **data_frame_index** – the list of columns that are used as DataFrame index

Returns the queried data

Return type *ResultSet*

write_points(*dataframe*, *measurement*, *tags=None*, *tag_columns=None*, *field_columns=None*, *time_precision=None*, *database=None*, *retention_policy=None*, *batch_size=None*, *protocol=u'line'*, *numeric_precision=None*)

Write to multiple time series names.

Parameters

- **dataframe** – data points in a DataFrame
- **measurement** – name of measurement
- **tags** – dictionary of tags, with string key-values
- **tag_columns** – [Optional, default None] List of data tag names
- **field_columns** – [Options, default None] List of data field names
- **time_precision** – [Optional, default None] Either 's', 'ms', 'u' or 'n'.
- **batch_size** (*int*) – [Optional] Value to write the points in batches instead of all at one time. Useful for when doing data dumps from one database to another or when doing a massive write operation
- **protocol** – Protocol for writing data. Either 'line' or 'json'.
- **numeric_precision** – Precision for floating point values. Either None, 'full' or some int, where int is the desired decimal precision. 'full' preserves full precision for int and float datatypes. Defaults to None, which preserves 14-15 significant figures for float and all significant figures for int datatypes.

1.2.3 SeriesHelper

class influxdb.**SeriesHelper**(***kw*)

Subclass this helper eases writing data points in bulk.

All data points are immutable, ensuring they do not get overwritten. Each subclass can write to its own database. The time series names can also be based on one or more defined fields. The field "time" can be specified when creating a point, and may be any of the time types supported by the client (i.e. str, datetime, int). If the time is not specified, the current system time (utc) will be used.

Annotated example:

```
class MySeriesHelper(SeriesHelper):
    class Meta:
        # Meta class stores time series helper configuration.
        series_name = 'events.stats.{server_name}'
        # Series name must be a string, curly brackets for dynamic use.
        fields = ['time', 'server_name']
        # Defines all the fields in this time series.
        ### Following attributes are optional. ###
        client = TestSeriesHelper.client
        # Client should be an instance of InfluxDBClient.
        :warning: Only used if autocommit is True.
        bulk_size = 5
        # Defines the number of data points to write simultaneously.
        # Only applicable if autocommit is True.
        autocommit = True
        # If True and no bulk_size, then will set bulk_size to 1.
        retention_policy = 'your_retention_policy'
        # Specify the retention policy for the data points
        time_precision = "h"|"m"|s"|"ms"|"u"|"ns"
        # Default is ns (nanoseconds)
        # Setting time precision while writing point
        # You should also make sure time is set in the given precision
```

classmethod commit(*client=None*)

Commit everything from datapoints via the client.

Parameters **client** – InfluxDBClient instance for writing points to InfluxDB.

Attention any provided client will supersede the class client.

Returns result of client.write_points.

1.2.4 ResultSet

See the *Query response object: ResultSet* page for more information.

class influxdb.resultset.**ResultSet**(*series*, *raise_errors=True*)

A wrapper around a single InfluxDB query result.

error

Error returned by InfluxDB.

get_points(*measurement=None*, *tags=None*)

Return a generator for all the points that match the given filters.

Parameters

- **measurement** (*str*) – The measurement name

- **tags** (*dict*) – Tags to look for

Returns Points generator

items()
Return the set of items from the ResultSet.

Returns List of tuples, (key, generator)

keys()
Return the list of keys in the ResultSet.

Returns List of keys. Keys are tuples (series_name, tags)

static point_from_cols_vals(*cols*, *vals*)
Create a dict from columns and values lists.

Parameters

- **cols** – List of columns
- **vals** – List of values

Returns Dict where keys are columns.

raw
Raw JSON from InfluxDB.

1.3 Exceptions

class influxdb.exceptions.**InfluxDBClientError**(*content*, *code=None*)
Raised when an error occurs in the request.

class influxdb.exceptions.**InfluxDBServerError**(*content*)
Raised when a server error occurs.

1.4 Query response object: ResultSet

Using the `InfluxDBClient.query()` function will return a `ResultSet` Object.

A ResultSet can be browsed in several ways. Its `get_points` method can be used to retrieve points generators that filter either by measurement, tags, or both.

1.4.1 Getting all points

Using `rs.get_points()` will return a generator for all the points in the ResultSet.

1.4.2 Filtering by measurement

Using `rs.get_points('cpu')` will return a generator for all the points that are in a series with measurement name `cpu`, no matter the tags.

```
rs = cli.query("SELECT * from cpu")
cpu_points = list(rs.get_points(measurement='cpu'))
```

1.4.3 Filtering by tags

Using `rs.get_points(tags={'host_name': 'influxdb.com'})` will return a generator for all the points that are tagged with the specified tags, no matter the measurement name.

```
rs = cli.query("SELECT * from cpu")
cpu_influxdb_com_points = list(rs.get_points(tags={"host_name": "influxdb.com"}))
```

1.4.4 Filtering by measurement and tags

Using measurement name and tags will return a generator for all the points that are in a series with the specified measurement name AND whose tags match the given tags.

```
rs = cli.query("SELECT * from cpu")
points = list(rs.get_points(measurement='cpu', tags={'host_name': 'influxdb.com'}))
```

See the *API Documentation* page for more information.

1.5 InfluxDB Python Examples

1.5.1 Tutorials - Basic

```
# -*- coding: utf-8 -*-
"""Tutorial on using the InfluxDB client."""

import argparse

from influxdb import InfluxDBClient


def main(host='localhost', port=8086):
    """Instantiate a connection to the InfluxDB."""
    user = 'root'
    password = 'root'
    dbname = 'example'
    dbuser = 'smly'
    dbuser_password = 'my_secret_password'
    query = 'select Float_value from cpu_load_short;'
    query_where = 'select Int_value from cpu_load_short where host=$host;'
    bind_params = {'host': 'server01'}
    json_body = [
        {
            "measurement": "cpu_load_short",
            "tags": {
                "host": "server01",
                "region": "us-west"
            },
            "time": "2009-11-10T23:00:00Z",
            "fields": {
                "Float_value": 0.64,
                "Int_value": 3,
                "String_value": "Text",
                "Bool_value": True
```

```
            }
        }
    ]

    client = InfluxDBClient(host, port, user, password, dbname)

    print("Create database: " + dbname)
    client.create_database(dbname)

    print("Create a retention policy")
    client.create_retention_policy('awesome_policy', '3d', 3, default=True)

    print("Switch user: " + dbuser)
    client.switch_user(dbuser, dbuser_password)

    print("Write points: {0}".format(json_body))
    client.write_points(json_body)

    print("Querying data: " + query)
    result = client.query(query)

    print("Result: {0}".format(result))

    print("Querying data: " + query_where)
    result = client.query(query_where, bind_params=bind_params)

    print("Result: {0}".format(result))

    print("Switch user: " + user)
    client.switch_user(user, password)

    print("Drop database: " + dbname)
    client.drop_database(dbname)


def parse_args():
    """Parse the args."""
    parser = argparse.ArgumentParser(
        description='example code to play with InfluxDB')
    parser.add_argument('--host', type=str, required=False,
                        default='localhost',
                        help='hostname of InfluxDB http API')
    parser.add_argument('--port', type=int, required=False, default=8086,
                        help='port of InfluxDB http API')
    return parser.parse_args()


if __name__ == '__main__':
    args = parse_args()
    main(host=args.host, port=args.port)
```

1.5.2 Tutorials - pandas

```
# -*- coding: utf-8 -*-
"""Tutorial for using pandas and the InfluxDB client."""
```

```
import argparse
import pandas as pd

from influxdb import DataFrameClient


def main(host='localhost', port=8086):
    """Instantiate the connection to the InfluxDB client."""
    user = 'root'
    password = 'root'
    dbname = 'demo'
    protocol = 'line'

    client = DataFrameClient(host, port, user, password, dbname)

    print("Create pandas DataFrame")
    df = pd.DataFrame(data=list(range(30)),
                      index=pd.date_range(start='2014-11-16',
                                          periods=30, freq='H'), columns=['0'])

    print("Create database: " + dbname)
    client.create_database(dbname)

    print("Write DataFrame")
    client.write_points(df, 'demo', protocol=protocol)

    print("Write DataFrame with Tags")
    client.write_points(df, 'demo',
                        {'k1': 'v1', 'k2': 'v2'}, protocol=protocol)

    print("Read DataFrame")
    client.query("select * from demo")

    print("Delete database: " + dbname)
    client.drop_database(dbname)


def parse_args():
    """Parse the args from main."""
    parser = argparse.ArgumentParser(
        description='example code to play with InfluxDB')
    parser.add_argument('--host', type=str, required=False,
                        default='localhost',
                        help='hostname of InfluxDB http API')
    parser.add_argument('--port', type=int, required=False, default=8086,
                        help='port of InfluxDB http API')
    return parser.parse_args()


if __name__ == '__main__':
    args = parse_args()
    main(host=args.host, port=args.port)
```

1.5.3 Tutorials - SeriesHelper

```
# -*- coding: utf-8 -*-
"""Tutorial how to use the class helper `SeriesHelper`."""

from influxdb import InfluxDBClient
from influxdb import SeriesHelper

# InfluxDB connections settings
host = 'localhost'
port = 8086
user = 'root'
password = 'root'
dbname = 'mydb'

myclient = InfluxDBClient(host, port, user, password, dbname)

# Uncomment the following code if the database is not yet created
# myclient.create_database(dbname)
# myclient.create_retention_policy('awesome_policy', '3d', 3, default=True)


class MySeriesHelper(SeriesHelper):
    """Instantiate SeriesHelper to write points to the backend."""

    class Meta:
        """Meta class stores time series helper configuration."""

        # The client should be an instance of InfluxDBClient.
        client = myclient

        # The series name must be a string. Add dependent fields/tags
        # in curly brackets.
        series_name = 'events.stats.{server_name}'

        # Defines all the fields in this time series.
        fields = ['some_stat', 'other_stat']

        # Defines all the tags for the series.
        tags = ['server_name']

        # Defines the number of data points to store prior to writing
        # on the wire.
        bulk_size = 5

        # autocommit must be set to True when using bulk_size
        autocommit = True


# The following will create *five* (immutable) data points.
# Since bulk_size is set to 5, upon the fifth construction call, *all* data
# points will be written on the wire via MySeriesHelper.Meta.client.
MySeriesHelper(server_name='us.east-1', some_stat=159, other_stat=10)
MySeriesHelper(server_name='us.east-1', some_stat=158, other_stat=20)
MySeriesHelper(server_name='us.east-1', some_stat=157, other_stat=30)
MySeriesHelper(server_name='us.east-1', some_stat=156, other_stat=30)
MySeriesHelper(server_name='us.east-1', some_stat=156)
MySeriesHelper(server_name='us.east-1', some_stat=155, other_stat=50)
```

```
# To manually submit data points which are not yet written, call commit:
MySeriesHelper.commit()

# To inspect the JSON which will be written, call _json_body_():
MySeriesHelper._json_body_()
```

1.5.4 Tutorials - UDP

```
# -*- coding: utf-8 -*-
"""Example for sending batch information to InfluxDB via UDP."""

"""
INFO: In order to use UDP, one should enable the UDP service from the
`influxdb.conf` under section
    [[udp]]
        enabled = true
        bind-address = ":8089" # port number for sending data via UDP
        database = "udp1" # name of database to be stored
    [[udp]]
        enabled = true
        bind-address = ":8090"
        database = "udp2"
"""


import argparse

from influxdb import InfluxDBClient


def main(uport):
    """Instantiate connection to the InfluxDB."""
    # NOTE: structure of the UDP packet is different than that of information
    #       sent via HTTP
    json_body = {
        "tags": {
            "host": "server01",
            "region": "us-west"
        },
        "points": [{
            "measurement": "cpu_load_short",
            "fields": {
                "value": 0.64
            },
            "time": "2009-11-10T23:00:00Z",
        },
        {
            "measurement": "cpu_load_short",
            "fields": {
                "value": 0.67
            },
            "time": "2009-11-10T23:05:00Z"
        }]
    }
```

```
    # make `use_udp` True and  add `udp_port` number from `influxdb.conf` file
    # no need to mention the database name since it is already configured
    client = InfluxDBClient(use_udp=True, udp_port=uport)

    # Instead of `write_points` use `send_packet`
    client.send_packet(json_body)


def parse_args():
    """Parse the args."""
    parser = argparse.ArgumentParser(
        description='example code to play with InfluxDB along with UDP Port')
    parser.add_argument('--uport', type=int, required=True,
                        help=' UDP port of InfluxDB')
    return parser.parse_args()


if __name__ == '__main__':
    args = parse_args()
    main(uport=args.uport)
```

1.5.5 Tutorials - Authorization by Token

```
# -*- coding: utf-8 -*-
"""Tutorial how to authorize InfluxDB client by custom Authorization token."""

import argparse
from influxdb import InfluxDBClient


def main(token='my-token'):
    """Instantiate a connection to the InfluxDB."""
    client = InfluxDBClient(username=None, password=None,
                            headers={"Authorization": token})

    print("Use authorization token: " + token)

    version = client.ping()
    print("Successfully connected to InfluxDB: " + version)
    pass


def parse_args():
    """Parse the args from main."""
    parser = argparse.ArgumentParser(
        description='example code to play with InfluxDB')
    parser.add_argument('--token', type=str, required=False,
                        default='my-token',
                        help='Authorization token for the proxy that is ahead the␣
↪InfluxDB.')
    return parser.parse_args()


if __name__ == '__main__':
    args = parse_args()
    main(token=args.token)
```

CHAPTER 2

Indices and tables

- genindex
- search

Index

W