Inflow Documentation

Release 0.2.1

Jaap Broekhuizen

Feb 24, 2017

Contents

A simple InfluxDB Python client library. It is an alternative for the official InfluxDB Python client library.

Inflow officially supports Python 2.7 and up, but the latest Python 3 version is recommended.

InfluxDB is supported from version 1.0 and up.

Documentation is hosted on Read the Docs.

Source code can be found on GitHub.

> **Warning:** This project is still very much in development, stuff might work, or not. API's might change, or even be removed. So be careful. This message will be removed once a stable version is released.

CHAPTER 1

Example

You can write measurements in a few different ways, but writing a single "temperature" measurement is as simple as:

```
from inflow import Client
client = Client('http://username:pass@localhost:8086/databasename')
client.write('temperature', value=21.3)
```

For more examples and docs on how to use the client, go to *Writing Measurements* and *Querying*.

CHAPTER 2

Installing

```
$ pip install inflow
```

CHAPTER 3

License

CHAPTER 4

Table of Contents

Writing Measurements

Examples

You can write measurements in a few different ways, but writing a single "temperature" measurement is as simple as:

```
#!/usr/bin/env python

from inflow import Client
client = Client('http://username:pass@localhost:8086/databasename')
client.write('temperature', value=21.3)
```

No time is specified in the above example, so inflow automatically set's the measurement's time to the current time. Also, no tags are provided to the `write` method, so no tags are attached to the measurement.

A more complex example of writing a single measurement:

```
#!/usr/bin/env python

from inflow import Client

client = Client('http://username:pass@localhost:8086/databasename')

client.write(
    'temperature',
    tags={
        'location': 'groningen',
        'sensor_type': 'ni1000'
    },
    value=21.3,
    timestamp=1475845863
)
```

Writing multiple measurements is also possible:

```
#!/usr/bin/env python

from inflow import Client, Measurement

client = Client('http://username:pass@localhost:8086/databasename')
```

```
client.write([
    Measurement(
        name='temperature',
        tags={
            'location': 'groningen',
            'sensor_type': 'ni1000'
        },
        value=21.3,
        timestamp=1475845863
    ),
    Measurement(
        name='temperature',
        tags={
            'location': 'groningen',
            'sensor_type': 'ni1000'
        },
        value=20.1,
        timestamp=1475848864
    )
])
```

However, this is a bit verbose. That's why you can also do this:

```
#!/usr/bin/env python

from inflow import Client

client = Client('http://username:pass@localhost:8086/databasename')

temperature = dict(
    name='temperature',
    tags={
        'location': 'groningen',
        'sensor_type': 'ni1000'
    }
)

client.write(temperature, [
    {'value': 21.3, 'timestamp': 1475845863},
    {'value': 20.1, 'timestamp': 1475846182}
])
```

In the above examples, every `write` call will issue a direct call to the InfluxDB API. You can accumulate measurements and write them all at once using *Sessions*.

Note: In every example, we use timestamp ints (in seconds) to specify the time for each measurement. You can also set the timestamp to a datetime. Inflow will automatically convert both to the right precision when writing to InfluxDB.

Warning: If you supply a Python *datetime* instance as the timestamp, make sure it is a timezone-aware instance, in the UTC timezone.

Multiple Values

In all the examples above, we assume there is only one actual `value` for the given measurements. However, InfluxDB supports having an arbitrary amount of values for every measurements. This is also possible in Inflow:

```
#!/usr/bin/env python

from inflow import Client

client = Client('http://username:pass@localhost:8086/databasename')

client.write(
    'temperature',
    timestamp=1475846182,
    lower_sensor=20.9,
    upper_sensor=23.2
)
```

This will create a measurement with the `lower_sensor` and `upper_sensor` values. This method also works when manually writing `Measurement` instances, and when writing lists of dicts.

Precision

By default, Inflow assumes the timestamps that are written to InfluxDB are in seconds. However, you can specify a custom precision when creating the client:

```
#!/usr/bin/env python

from inflow import Client
client = Client('http://username:pass@localhost:8086/databasename',
                precision='ms')
client.write('temperature', value=21.3, timestamp=1476191999000)
```

The precision needs to be one of: *h*, *m*, *s*, *ms*, *u* or *ns*.

Retention Policies

By default, Inflow will write to the database's default retention policy. However, you can explicitly specify which retention policy your measurements should be written to:

```
#!/usr/bin/env python

from inflow import Client

client = Client('http://username:pass@localhost:8086/databasename',
                retention_policy='rp_four_weeks')

client.write('temperature', value=21.3)
```

You can also specify the retention policy when calling into `write`:

```
#!/usr/bin/env python

from inflow import Client
client = Client('http://username:pass@localhost:8086/databasename')
client.write('temperature', value=21.3, retention_policy='rp_four_weeks')
```

Sessions

In the examples listed in *Writing Measurements*, every call to `write` will issue a direct call to the InfluxDB API. There might be situations where you'd want to accumulate measurements and write them all at once. That's what sessions are for:

```
from inflow import Client

client = Client('http://username:pass@localhost:8086/databasename')
session = client.session()

session.write('temperature', value=23.1, timestamp=1475848864)
session.write('temperature', value=25.0, timestamp=1475849823)

session.commit()
```

In the above example, a session is created in which we issue our `write` calls. After doing some `write` calls, we call `commit` on the session. This will issue the write to the InfluxDB API. If `commit` isn't called on the sessions, the data given in the `write`'s will be lost.

Note: The session's `write` method works exactly the same as that of the normal client.

Warning: Don't try to call `query` on a session, as Sessions are only meant to do writes. If you want to do queries, just use the Client.

As a Context Manager

You can also use the session as a context manager:

```
from inflow import Client

client = Client('http://username:pass@localhost:8086/databasename')

with client.session() as session:
    session.write('temperature', value=23.1, timestamp=1475848864)
    session.write('temperature', value=25.0, timestamp=1475849823)
```

When the context manager exits, the session is automatically committed.

Autocommitting

You can also have the session autocommit after a certain amount of `write` calls, using `autocommit_every`:

```
from inflow import Client

client = Client('http://username:pass@localhost:8086/databasename')
session = client.session(autocommit_every=5)
```

```
session.write('temperature', value=23.1, timestamp=1475848864)
session.write('temperature', value=25.0, timestamp=1475849823)
session.write('temperature', value=22.9, timestamp=1475849825)
session.write('temperature', value=28.2, timestamp=1475849912)

# This next write call will trigger the autocommit.
session.write('temperature', value=25.1, timestamp=1475849999)
```

Retention Policies

You can also specify the retention policy for the entire session:

```
from inflow import Client

client = Client('http://username:pass@localhost:8086/databasename')

with client.session(retention_policy='rp_four_weeks') as session:
    session.write('temperature', value=23.1, timestamp=1475848864)
```

Note: Unlike the *Client.write* method, you cannot specify the retention policy on the *Session.write*. Retention policies are Session-wide.

Querying

Inflow contains a minimal abstraction over the `/query` endpoint of the InfluxDB HTTP API.

Getting a list of measurements is as simple as:

```
#!/usr/bin/env python

from inflow import Client

client = Client('http://username:pass@localhost:8086/databasename')
results = client.query('SELECT * FROM "temperatures"')
# Results will contain a list of dicts for each returned measurement.
```

Say you've got a measurement called `temperature` (just as in the example above), which contains a `value` field, a `location` tag, and contains 2 values. To query that data you would call the `query` method as described in the above example, which will return a list with the following structure:

```
[
    {
        'name': 'temperature',
        'values': [
            {
                'value': 21.0,
                'timestamp': '2016-01-10T00:01:00Z',
                'location': 'groningen'
            },
            {
                'valuestamp': 23.0,
                'time': '2016-01-10T00:02:00Z',
```

```
                'location': 'groningen'
            }
        ]
    }
]
```

You can use any query type that InfluxDB allows, and it should work.

Unix Timestamps

By default, InfluxDB will return timestamps in RFC3339 format with nanosecond precision. If you want instead want unix timestamps (in a specific precision), you can use the `epoch` kwarg, like this:

```
#!/usr/bin/env python

from inflow import Client

client = Client('http://username:pass@localhost:8086/databasename')
results = client.query('SELECT * FROM "temperatures"', epoch='s')
```

In this example, we specify that we want unix timestamps, in seconds. The `epoch` argument accepts one of `h`, `m`, `s`, `ms`, `u` and `ns`.

Error Handling

If you're doing something that the InfluxDB API deems wrong, it will return an error. These errors can occur when trying to query or write data. The exceptions described below wrap the errors returned by the InfluxDB API, and you should probably make sure you handle them in your code.

Exception types

class `inflow.`**`InfluxDBException`**
Generic exception for InfluxDB HTTP error's, all other exceptions subclass from this one.

The message in this exception (and it's subclasses) is the raw error message returned by the InfluxDB HTTP API.

class `inflow.`**`QueryFailedException`**
Thrown when a query is rejected by the API. For example, this happens when you have a syntactically incorrect query.

class `inflow.`**`WriteFailedException`**
Thrown when a write is rejected by the API.

class `inflow.`**`DatabaseNotFoundException`**
Thrown when trying to write to a non-existing database.

class `inflow.`**`UnauthorizedException`**
Thrown when trying to log in using incorrect credentials.

class `inflow.`**`ForbiddenException`**
Thrown when a user is correctly logged in, but is not allowed to do the query or write action it wants to do.