asphalt-influxdb

Release 2.1.0.post3

Oct 01, 2017

Contents

This Asphalt framework component provides connectivity to InfluxDB database servers.

CHAPTER 1

Configuration

The typical InfluxDB configuration using a single database at `localhost` on the default port would look like this:

```
components:
  influxdb:
    db: mydb
```

The above configuration creates an `asphalt.influxdb.client.InfluxDBClient` instance in the context, available as `ctx.influxdb` (resource name: `default`).

If you wanted to connect to `influx.example.org` on port 8886, you would do:

```
components:
  influxdb:
    base_urls: http://influx.example.org:8886
    db: mydb
```

To connect to an InfluxEnterprise cluster, list all the nodes under `base_urls`:

```
components:
  influxdb:
    base_urls:
      - http://influx1.example.org:8086
      - http://influx2.example.org:8086
      - http://influx3.example.org:8086
    db: mydb
```

To connect to two unrelated InfluxDB servers, you could use a configuration like:

```
components:
  influxdb:
    clients:
      influx1:
        base_urls: http://influx.example.org:8886
        db: mydb
      influx2:
```

```
        context_attr: influxalter
        base_urls: https://influxalter.example.org/influxdb
        db: anotherdb
        username: testuser
        password: testpass
```

This configuration creates two `asphalt.influxdb.client.InfluxDBClient` resources, `influx1` and `influx2` (`ctx.influx1` and `ctx.influxalter`) respectively.

Note: See the documentation of the `asphalt.influxdb.client.InfluxDBClient` class for a comprehensive listing of all connection options.

CHAPTER 2

Using the InfluxDB client

2.1 Getting the server version

You can find out which version of InfluxDB you're running in the following manner:

```
async def handler(ctx):
    version = await ctx.influxdb.ping()
    print('Running InfluxDB version ' + version)
```

2.2 Writing data points

Each data point being written into an InfluxDB database contains the following information:

- name of the measurement (corresponds to a table in a relational database)
- one or more tags (corresponds to indexed, non-nullable string columns in a RDBMS)
- zero or more fields (corresponds to nullable columns in a RDBMS)
- a timestamp (supplied by the server if omitted)

Data points can be written one at a time (`write()`) or many at once (`write_many()`). It should be noted that the former is merely a wrapper for the latter.

To write a single data point:

```
async def handler(ctx):
    await ctx.influxdb.write('cpu_load_short', dict(host=server02), dict(value=0.67))
    print('Running InfluxDB version ' + version)
```

To write multiple data points, you need use use the `DataPoint` class:

```
async def handler(ctx):
    await ctx.influxdb.write_many([
        DataPoint('cpu_load_short', dict(host=server02), dict(value=0.67)),
        DataPoint('cpu_load_short', dict(host=server01), dict(value=0.21))
    ])
```

The data points don't have to be within the same measurement.

2.3 Querying the database

This library offers the ability to query data via both raw InfluxQL queries and a programmatic query builder. To learn how the query builder works, see the next section.

Sending raw queries is done using `raw_query()`:

```
async def handler(ctx):
    series = await ctx.influxdb.raw_query('SHOW DATABASES')
    print([row[0] for row in series])
```

If you include more than one measurement in the `FROM` clause of a `SELECT` query, you will get a list of `Series` as a result:

```
async def handler(ctx):
    cpu_load, temperature = await ctx.influxdb.raw_query('SELECT * FROM cpu_load,
↪temperature')
    print([row[0] for row in series])
```

It is possible to send multiple queries by delimiting them with a semicolon (`;`). If you do that, the call will return a list of results for each query, each of which may be a `Series` or a list thereof:

```
async def handler(ctx):
    db_series, m_series = await ctx.influxdb.raw_query('SHOW DATABASES; SHOW␣
↪MEASUREMENTS')
    print('Databases:')
    print([row[0] for row in db_series])
    print('Measurements:')
    print([row[0] for row in m_series])
```

Warning: Due to this bug, multiple queries with the same call do not currently work.

Note: If you want to send a query like `SELECT ... INTO ...`, you must call `raw_query()` with `http_verb='POST'`. The proper HTTP verb should be automatically detected from the query string for all other queries.

2.4 Using the query builder

The query builder allows one to dynamically build queries without having to do error prone manual string concatenation. The query builder is considered immutable, so every one of its methods returns a new builder with the modifications made to it, just like with SQLAlchemy ORM queries.

For example, to select `field1` and `field2` from measurements `m1` and `m2`:

```
async def handler(ctx):
    query = ctx.influxdb.query(['field1', 'field2'], ['m1', 'm2'])
    return await query.execute()
```

This will produce a query like `SELECT "field1","field2" FROM "m1","m2"`.

More complex expressions can be given but field or tag names are not automatically quoted:

```
async def handler(ctx):
    query = ctx.influxdb.query('field1 + field2', 'm1')
    return await query.execute()
```

The query will look like `SELECT field1 + field2 FROM "m1"`.

Filters can be added by using `where()` with positional and/or keyword arguments:

```
async def handler(ctx):
    query = ctx.influxdb.query(['field1', 'field2'], 'm1').\
        where('field1 > 3.5', 'field2 < 6.2', location='Helsinki')
    return await query.execute()
```

This will produce a query like `SELECT field1,field2 FROM "m1" WHERE field1 > 3.5 AND field2 < 6.2 AND location='Helsinki'`.

To use `OR`, you have to manually include it in one of the `WHERE` expressions.

Further calls to `.where()` will add conditions to the `WHERE` clause of the query. A call to `.where()` with no arguments will clear the `WHERE` clause.

Grouping by tags works largely the same way:

```
async def handler(ctx):
    query = ctx.influxdb.query(['field1', 'SUM(field2)'], 'm1').group_by('tag1')
    return await query.execute()
```

The SQL: `SELECT field1,SUM(field2) FROM "m1" GROUP BY "tag1"`

CHAPTER 3

Version history

This library adheres to Semantic Versioning.

2.1.0 (2017-09-20)

- Exposed the `Series.values` attribute to enable faster processing of query results

2.0.1 (2017-06-04)

- Added compatibility with Asphalt 4.0
- Fixed `DeprecationWarning: ClientSession.close() is not coroutine` warnings
- Added Docker configuration for easier local testing

2.0.0 (2017-04-11)

- **BACKWARD INCOMPATIBLE** Migrated to Asphalt 3.0
- **BACKWARD INCOMPATIBLE** Migrated to aiohttp 2.0

1.1.1 (2017-02-09)

- Fixed handling of long responses (on InfluxDB 1.2+)

1.1.0 (2016-12-15)

- Added the `KeyedTuple._asdict()` method
- Fixed wrong quoting of string values (should use single quotes, not double quotes)

1.0.0 (2016-12-12)

- Initial release
- API reference