influxdb$_{client}$

Release 1.26.0

Feb 18, 2022

Contents:

CHAPTER 1

User Guide

- *Efficiency write data from IOT sensor*
- *Connect to InfluxDB Cloud*
- *How to use Jupyter + Pandas + InfluxDB 2*
- *Other examples*

1.1 Query

```
from influxdb_client import InfluxDBClient, Point
from influxdb_client.client.write_api import SYNCHRONOUS

bucket = "my-bucket"

client = InfluxDBClient(url="http://localhost:8086", token="my-token", org="my-org")

write_api = client.write_api(write_options=SYNCHRONOUS)
query_api = client.query_api()

p = Point("my_measurement").tag("location", "Prague").field("temperature", 25.3)

write_api.write(bucket=bucket, record=p)

## using Table structure
tables = query_api.query('from(bucket:"my-bucket") |> range(start: -10m)')

for table in tables:
    print(table)
    for row in table.records:
        print (row.values)


## using csv library
csv_result = query_api.query_csv('from(bucket:"my-bucket") |> range(start: -10m)')
val_count = 0
for row in csv_result:
    for cell in row:
        val_count += 1
```

1.2 Write

The WriteApi supports synchronous, asynchronous and batching writes into InfluxDB 2.0. The data should be passed as a InfluxDB Line Protocol, Data Point or Observable stream.

Warning: The `WriteApi` in batching mode (default mode) is suppose to run as a singleton. To flush all your data you should wrap the execution using `with client.write_api(...) as write_api:` statement or call `write_api.close()` at the end of your script.

The default instance of WriteApi use batching.

1.2.1 The data could be written as

1. `string` or `bytes` that is formatted as a InfluxDB's line protocol
2. Data Point structure
3. Dictionary style mapping with keys: `measurement`, `tags`, `fields` and `time` or custom structure
4. NamedTuple
5. Data Classes
6. Pandas DataFrame
7. List of above items
8. A `batching` type of write also supports an `Observable` that produce one of an above item

You can find write examples at GitHub: influxdb-client-python/examples.

1.2.2 Batching

The batching is configurable by `write_options`:

Property	Description	Default Value
batch_size	the number of data pointx to collect in a batch	`1000`
flush_interval	the number of milliseconds before the batch is written	`1000`
jitter_interval	the number of milliseconds to increase the batch flush interval by a random amount	`0`
retry_interval	the number of milliseconds to retry first unsuccessful write. The next retry delay is computed using exponential random backoff. The retry interval is used when the InfluxDB server does not specify "Retry-After" header.	`5000`
max_retry_time	maximum total retry timeout in milliseconds.	`180_000`
max_retries	the number of max retries when write fails	`5`
max_retry_delay	the maximum delay between each retry attempt in milliseconds	`125_000`
exponential_base	the base for the exponential retry delay, the next delay is computed using random exponential backoff as a random value within the interval `retry_interval * exponential_base^(attempts-1)` and `retry_interval * exponential_base^(attempts)`. Example for `retry_interval=5_000, exponential_base=2, max_retry_delay=125_000, total=5` Retry delays are random distributed values within the ranges of `[5_000-10_000, 10_000-20_000, 20_000-40_000, 40_000-80_000, 80_000-125_000]`	`2`

```
from datetime import datetime, timedelta

import pandas as pd
import rx
from pytz import UTC
from rx import operators as ops

from influxdb_client import InfluxDBClient, Point, WriteOptions

with InfluxDBClient(url="http://localhost:8086", token="my-token", org="my-org") as _
↪client:
```

(continues on next page)

(continued from previous page)

```
    with _client.write_api(write_options=WriteOptions(batch_size=500,
                                                      flush_interval=10_000,
                                                      jitter_interval=2_000,
                                                      retry_interval=5_000,
                                                      max_retries=5,
                                                      max_retry_delay=30_000,
                                                      exponential_base=2)) as _write_
↪client:

        """
        Write Line Protocol formatted as string
        """
        _write_client.write("my-bucket", "my-org", "h2o_feet,location=coyote_creek␣
↪water_level=1.0 1")
        _write_client.write("my-bucket", "my-org", ["h2o_feet,location=coyote_creek␣
↪water_level=2.0 2",
                                                    "h2o_feet,location=coyote_creek␣
↪water_level=3.0 3"])

        """
        Write Line Protocol formatted as byte array
        """
        _write_client.write("my-bucket", "my-org", "h2o_feet,location=coyote_creek␣
↪water_level=1.0 1".encode())
        _write_client.write("my-bucket", "my-org", ["h2o_feet,location=coyote_creek␣
↪water_level=2.0 2".encode(),
                                                    "h2o_feet,location=coyote_creek␣
↪water_level=3.0 3".encode()])

        """
        Write Dictionary-style object
        """
        _write_client.write("my-bucket", "my-org", {"measurement": "h2o_feet", "tags
↪": {"location": "coyote_creek"},
                                                    "fields": {"water_level": 1.0},
↪"time": 1})
        _write_client.write("my-bucket", "my-org", [{"measurement": "h2o_feet", "tags
↪": {"location": "coyote_creek"},
                                                     "fields": {"water_level": 2.0},
↪"time": 2},
                                                    {"measurement": "h2o_feet", "tags
↪": {"location": "coyote_creek"},
                                                     "fields": {"water_level": 3.0},
↪"time": 3}])

        """
        Write Data Point
        """
        _write_client.write("my-bucket", "my-org",
                            Point("h2o_feet").tag("location", "coyote_creek").field(
↪"water_level", 4.0).time(4))
        _write_client.write("my-bucket", "my-org",
                            [Point("h2o_feet").tag("location", "coyote_creek").field(
↪"water_level", 5.0).time(5),
                             Point("h2o_feet").tag("location", "coyote_creek").field(
↪"water_level", 6.0).time(6)])
```

(continues on next page)

(continued from previous page)

```
        """
        Write Observable stream
        """
        _data = rx \
            .range(7, 11) \
            .pipe(ops.map(lambda i: "h2o_feet,location=coyote_creek water_level={0}.0
↪{0}".format(i)))

        _write_client.write("my-bucket", "my-org", _data)

        """
        Write Pandas DataFrame
        """
        _now = datetime.now(UTC)
        _data_frame = pd.DataFrame(data=[["coyote_creek", 1.0], ["coyote_creek", 2.
↪0]],
                                   index=[_now, _now + timedelta(hours=1)],
                                   columns=["location", "water_level"])

        _write_client.write("my-bucket", "my-org", record=_data_frame, data_frame_
↪measurement_name='h2o_feet',
                            data_frame_tag_columns=['location'])
```

1.2.3 Default Tags

Sometimes is useful to store same information in every measurement e.g. `hostname`, `location`, `customer`. The client is able to use static value or env property as a tag value.

The expressions:

- `California Miner` - static value
- `${env.hostname}` - environment property

Via API

```
point_settings = PointSettings()
point_settings.add_default_tag("id", "132-987-655")
point_settings.add_default_tag("customer", "California Miner")
point_settings.add_default_tag("data_center", "${env.data_center}")

self.write_client = self.client.write_api(write_options=SYNCHRONOUS, point_
↪settings=point_settings)
```

```
self.write_client = self.client.write_api(write_options=SYNCHRONOUS,
                                          point_settings=PointSettings(**{"id":
↪"132-987-655",

↪"customer": "California Miner"}))
```

Via Configuration file

In a init configuration file you are able to specify default tags by `tags` segment.

```
self.client = InfluxDBClient.from_config_file("config.ini")
```

You can also use a TOML format for the configuration file.

Via Environment Properties

You are able to specify default tags by environment properties with prefix `INFLUXDB_V2_TAG_`.

Examples:

- `INFLUXDB_V2_TAG_ID`
- `INFLUXDB_V2_TAG_HOSTNAME`

```
self.client = InfluxDBClient.from_env_properties()
```

1.2.4 Asynchronous client

Data are writes in an asynchronous HTTP request.

```
from influxdb_client import InfluxDBClient, Point
from influxdb_client.client.write_api import ASYNCHRONOUS

client = InfluxDBClient(url="http://localhost:8086", token="my-token", org="my-org")
write_api = client.write_api(write_options=ASYNCHRONOUS)

_point1 = Point("my_measurement").tag("location", "Prague").field("temperature", 25.3)
_point2 = Point("my_measurement").tag("location", "New York").field("temperature", 24.
↪3)

async_result = write_api.write(bucket="my-bucket", record=[_point1, _point2])
async_result.get()

client.close()
```

1.2.5 Synchronous client

Data are writes in a synchronous HTTP request.

```
from influxdb_client import InfluxDBClient, Point
from influxdb_client .client.write_api import SYNCHRONOUS

client = InfluxDBClient(url="http://localhost:8086", token="my-token", org="my-org")
write_api = client.write_api(write_options=SYNCHRONOUS)

_point1 = Point("my_measurement").tag("location", "Prague").field("temperature", 25.3)
_point2 = Point("my_measurement").tag("location", "New York").field("temperature", 24.
↪3)

write_api.write(bucket="my-bucket", record=[_point1, _point2])

client.close()
```

1.3 Pandas DataFrame

Note: For DataFrame querying you should install Pandas dependency via `pip install 'influxdb-client[extra]'`.

Note: Note that if a query returns more then one table then the client generates a `DataFrame` for each of them.

The `client` is able to retrieve data in Pandas DataFrame format thought `query_data_frame`:

```
from influxdb_client import InfluxDBClient, Point, Dialect
from influxdb_client.client.write_api import SYNCHRONOUS

client = InfluxDBClient(url="http://localhost:8086", token="my-token", org="my-org")

write_api = client.write_api(write_options=SYNCHRONOUS)
query_api = client.query_api()

"""
Prepare data
"""

_point1 = Point("my_measurement").tag("location", "Prague").field("temperature", 25.3)
_point2 = Point("my_measurement").tag("location", "New York").field("temperature", 24.
↪3)

write_api.write(bucket="my-bucket", record=[_point1, _point2])

"""
Query: using Pandas DataFrame
"""
data_frame = query_api.query_data_frame('from(bucket:"my-bucket") '
                                        '|> range(start: -10m) '
                                        '|> pivot(rowKey:["_time"], columnKey: ["_
↪field"], valueColumn: "_value") '
                                        '|> keep(columns: ["location", "temperature"])
↪')
print(data_frame.to_string())

"""
Close client
"""
client.close()
```

Output:

1.4 Delete data

The delete_api.py supports deletes points from an InfluxDB bucket.

```
from influxdb_client import InfluxDBClient
```

(continues on next page)

(continued from previous page)

```
client = InfluxDBClient(url="http://localhost:8086", token="my-token")

delete_api = client.delete_api()

"""
Delete Data
"""
start = "1970-01-01T00:00:00Z"
stop = "2021-02-01T00:00:00Z"
delete_api.delete(start, stop, '_measurement="my_measurement"', bucket='my-bucket',␣
↪org='my-org')

"""
Close client
"""
client.close()
```

1.5 Gzip support

`InfluxDBClient` does not enable gzip compression for http requests by default. If you want to enable gzip to reduce transfer data's size, you can call:

```
from influxdb_client import InfluxDBClient

_db_client = InfluxDBClient(url="http://localhost:8086", token="my-token", org="my-org
↪", enable_gzip=True)
```

1.6 Proxy configuration

You can configure the client to tunnel requests through an HTTP proxy. The following proxy options are supported:

- `proxy` - Set this to configure the http proxy to be used, ex. `http://localhost:3128`
- `proxy_headers` - A dictionary containing headers that will be sent to the proxy. Could be used for proxy authentication.

```
from influxdb_client import InfluxDBClient

with InfluxDBClient(url="http://localhost:8086",
                    token="my-token",
                    org="my-org",
                    proxy="http://localhost:3128") as client:
```

Note: If your proxy notify the client with permanent redirect (`HTTP 301`) to **different host**. The client removes `Authorization` header, because otherwise the contents of `Authorization` is sent to third parties which is a security vulnerability.

You can change this behaviour by:

```
from urllib3 import Retry
Retry.DEFAULT_REMOVE_HEADERS_ON_REDIRECT = frozenset()
Retry.DEFAULT.remove_headers_on_redirect = Retry.DEFAULT_REMOVE_HEADERS_ON_REDIRECT
```

1.7 Nanosecond precision

The Python's datetime doesn't support precision with nanoseconds so the library during writes and queries ignores everything after microseconds.

If you would like to use `datetime` with nanosecond precision you should use pandas.Timestamp that is replacement for python `datetime.datetime` object and also you should set a proper `DateTimeHelper` to the client.

- sources - nanosecond_precision.py

```
from influxdb_client import Point, InfluxDBClient
from influxdb_client.client.util.date_utils_pandas import PandasDateTimeHelper
from influxdb_client.client.write_api import SYNCHRONOUS

"""
Set PandasDate helper which supports nanoseconds.
"""
import influxdb_client.client.util.date_utils as date_utils

date_utils.date_helper = PandasDateTimeHelper()

"""
Prepare client.
"""
client = InfluxDBClient(url="http://localhost:8086", token="my-token", org="my-org")

write_api = client.write_api(write_options=SYNCHRONOUS)
query_api = client.query_api()

"""
Prepare data
"""

point = Point("h2o_feet") \
    .field("water_level", 10) \
    .tag("location", "pacific") \
    .time('1996-02-25T21:20:00.001001231Z')

print(f'Time serialized with nanosecond precision: {point.to_line_protocol()}')
print()

write_api.write(bucket="my-bucket", record=point)

"""
Query: using Stream
"""
query = '''
from(bucket:"my-bucket")
        |> range(start: 0, stop: now())
        |> filter(fn: (r) => r._measurement == "h2o_feet")
'''
records = query_api.query_stream(query)

for record in records:
```

(continues on next page)

(continued from previous page)

```
    print(f'Temperature in {record["location"]} is {record["_value"]} at time:
↪{record["_time"]}')

"""
Close client
"""
client.close()
```

1.8 Handling Errors

Errors happen and it's important that your code is prepared for them. All client related exceptions are delivered from `InfluxDBError`. If the exception cannot be recovered in the client it is returned to the application. These exceptions are left for the developer to handle.

Almost all APIs directly return unrecoverable exceptions to be handled this way:

```
from influxdb_client import InfluxDBClient
from influxdb_client.client.exceptions import InfluxDBError
from influxdb_client.client.write_api import SYNCHRONOUS

with InfluxDBClient(url="http://localhost:8086", token="my-token", org="my-org") as␣
↪client:
    try:
        client.write_api(write_options=SYNCHRONOUS).write("my-bucket", record="mem,
↪tag=a value=86")
    except InfluxDBError as e:
        if e.response.status == 401:
            raise Exception(f"Insufficient write permissions to 'my-bucket'.") from e
        raise
```

The only exception is **batching** `WriteAPI` (for more info see *Batching*). where you need to register custom callbacks to handle batch events. This is because this API runs in the `background` in a `separate` thread and isn't possible to directly return underlying exceptions.

```
from influxdb_client import InfluxDBClient
from influxdb_client.client.exceptions import InfluxDBError


class BatchingCallback(object):

    def success(self, conf: (str, str, str), data: str):
        print(f"Written batch: {conf}, data: {data}")

    def error(self, conf: (str, str, str), data: str, exception: InfluxDBError):
        print(f"Cannot write batch: {conf}, data: {data} due: {exception}")

    def retry(self, conf: (str, str, str), data: str, exception: InfluxDBError):
        print(f"Retryable error occurs for batch: {conf}, data: {data} retry:
↪{exception}")


with InfluxDBClient(url="http://localhost:8086", token="my-token", org="my-org") as␣
↪client:
    callback = BatchingCallback()
```

(continues on next page)

(continued from previous page)

```
    with client.write_api(success_callback=callback.success,
                          error_callback=callback.error,
                          retry_callback=callback.retry) as write_api:
        pass
```

1.8.1 HTTP Retry Strategy

By default the client uses a retry strategy only for batching writes (for more info see *Batching*). For other HTTP requests there is no one retry strategy, but it could be configured by `retries` parameter of `InfluxDBClient`.

For more info about how configure HTTP retry see details in urllib3 documentation.

```
from urllib3 import Retry

from influxdb_client import InfluxDBClient

retries = Retry(connect=5, read=2, redirect=5)
client = InfluxDBClient(url="http://localhost:8086", token="my-token", org="my-org",␣
↪retries=retries)
```

1.9 Debugging

For debug purpose you can enable verbose logging of http requests. Both request header and body will be logged to standard output.

```
_client = InfluxDBClient(url="http://localhost:8086", token="my-token", debug=True,␣
↪org="my-org")
```

1.10 Examples

1.10.1 How to efficiently import large dataset

The following example shows how to import dataset with dozen megabytes. If you would like to import gigabytes of data then use our multiprocessing example: import_data_set_multiprocessing.py for use a full capability of your hardware.

- sources - import_data_set.py

```
"""
Import VIX - CBOE Volatility Index - from "vix-daily.csv" file into InfluxDB 2.0

https://datahub.io/core/finance-vix#data
"""

from collections import OrderedDict
from csv import DictReader

import rx
from rx import operators as ops

```

(continues on next page)

(continued from previous page)

```
from influxdb_client import InfluxDBClient, Point, WriteOptions

def parse_row(row: OrderedDict):
    """Parse row of CSV file into Point with structure:

        financial-analysis,type=ily close=18.47,high=19.82,low=18.28,open=19.82␣
↪1198195200000000000

    CSV format:
        Date,VIX Open,VIX High,VIX Low,VIX Close\n
        2004-01-02,17.96,18.68,17.54,18.22\n
        2004-01-05,18.45,18.49,17.44,17.49\n
        2004-01-06,17.66,17.67,16.19,16.73\n
        2004-01-07,16.72,16.75,15.5,15.5\n
        2004-01-08,15.42,15.68,15.32,15.61\n
        2004-01-09,16.15,16.88,15.57,16.75\n
        ...

    :param row: the row of CSV file
    :return: Parsed csv row to [Point]
    """

    """
     For better performance is sometimes useful directly create a LineProtocol to␣
↪avoid unnecessary escaping overhead:
     """
     # from pytz import UTC
     # import ciso8601
     # from influxdb_client.client.write.point import EPOCH
     #
     # time = (UTC.localize(ciso8601.parse_datetime(row["Date"])) - EPOCH).total_
↪seconds() * 1e9
     # return f"financial-analysis,type=vix-daily" \
     #        f" close={float(row['VIX Close'])},high={float(row['VIX High'])},low=
↪{float(row['VIX Low'])},open={float(row['VIX Open'])} " \
     #        f" {int(time)}"

    return Point("financial-analysis") \
        .tag("type", "vix-daily") \
        .field("open", float(row['VIX Open'])) \
        .field("high", float(row['VIX High'])) \
        .field("low", float(row['VIX Low'])) \
        .field("close", float(row['VIX Close'])) \
        .time(row['Date'])


"""
Converts vix-daily.csv into sequence of datad point
"""
data = rx \
    .from_iterable(DictReader(open('vix-daily.csv', 'r'))) \
    .pipe(ops.map(lambda row: parse_row(row)))

client = InfluxDBClient(url="http://localhost:8086", token="my-token", org="my-org",␣
↪debug=True)

"""
```

(continues on next page)

(continued from previous page)

```
Create client that writes data in batches with 50_000 items.
"""
write_api = client.write_api(write_options=WriteOptions(batch_size=50_000, flush_
↪interval=10_000))

"""
Write data into InfluxDB
"""
write_api.write(bucket="my-bucket", record=data)
write_api.close()

"""
Querying max value of CBOE Volatility Index
"""
query = 'from(bucket:"my-bucket")' \
        ' |> range(start: 0, stop: now())' \
        ' |> filter(fn: (r) => r._measurement == "financial-analysis")' \
        ' |> max()'
result = client.query_api().query(query=query)

"""
Processing results
"""
print()
print("=== results ===")
print()
for table in result:
    for record in table.records:
        print('max {0:5} = {1}'.format(record.get_field(), record.get_value()))

"""
Close client
"""
client.close()
```

1.10.2 Efficiency write data from IOT sensor

- sources - iot_sensor.py

```
"""
Efficiency write data from IOT sensor - write changed temperature every minute
"""
import atexit
import platform
from datetime import timedelta

import psutil as psutil
import rx
from rx import operators as ops

from influxdb_client import InfluxDBClient, WriteApi, WriteOptions

def on_exit(db_client: InfluxDBClient, write_api: WriteApi):
    """Close clients after terminate a script.
```

(continues on next page)

(continued from previous page)

```
    :param db_client: InfluxDB client
    :param write_api: WriteApi
    :return: nothing
    """
    write_api.close()
    db_client.close()


def sensor_temperature():
    """Read a CPU temperature. The [psutil] doesn't support MacOS so we use [sysctl].

    :return: actual CPU temperature
    """
    os_name = platform.system()
    if os_name == 'Darwin':
        from subprocess import check_output
        output = check_output(["sysctl", "machdep.xcpm.cpu_thermal_level"])
        import re
        return re.findall(r'\d+', str(output))[0]
    else:
        return psutil.sensors_temperatures()["coretemp"][0]


def line_protocol(temperature):
    """Create a InfluxDB line protocol with structure:

        iot_sensor,hostname=mine_sensor_12,type=temperature value=68

    :param temperature: the sensor temperature
    :return: Line protocol to write into InfluxDB
    """

    import socket
    return 'iot_sensor,hostname={},type=temperature value={}'.format(socket.
↪gethostname(), temperature)


"""
Read temperature every minute; distinct_until_changed - produce only if temperature␣
↪change
"""
data = rx\
    .interval(period=timedelta(seconds=60))\
    .pipe(ops.map(lambda t: sensor_temperature()),
          ops.distinct_until_changed(),
          ops.map(lambda temperature: line_protocol(temperature)))

_db_client = InfluxDBClient(url="http://localhost:8086", token="my-token", org="my-org
↪", debug=True)

"""
Create client that writes data into InfluxDB
"""
_write_api = _db_client.write_api(write_options=WriteOptions(batch_size=1))
_write_api.write(bucket="my-bucket", record=data)
```

(continues on next page)

(continued from previous page)

```
"""
Call after terminate a script
"""
atexit.register(on_exit, _db_client, _write_api)

input()
```

1.10.3 Connect to InfluxDB Cloud

The following example demonstrate a simplest way how to write and query date with the InfluxDB Cloud.

At first point you should create an authentication token as is described here.

After that you should configure properties: `influx_cloud_url`, `influx_cloud_token`, `bucket` and `org` in a `influx_cloud.py` example.

The last step is run a python script via: `python3 influx_cloud.py`.

- sources - influx_cloud.py

```
"""
Connect to InfluxDB 2.0 - write data and query them
"""

from datetime import datetime

from influxdb_client import Point, InfluxDBClient
from influxdb_client.client.write_api import SYNCHRONOUS

"""
Configure credentials
"""
influx_cloud_url = 'https://us-west-2-1.aws.cloud2.influxdata.com'
influx_cloud_token = '...'
bucket = '...'
org = '...'

client = InfluxDBClient(url=influx_cloud_url, token=influx_cloud_token)
try:
    kind = 'temperature'
    host = 'host1'
    device = 'opt-123'

    """
    Write data by Point structure
    """
    point = Point(kind).tag('host', host).tag('device', device).field('value', 25.3).
↪time(time=datetime.utcnow())

    print(f'Writing to InfluxDB cloud: {point.to_line_protocol()} ...')

    write_api = client.write_api(write_options=SYNCHRONOUS)
    write_api.write(bucket=bucket, org=org, record=point)

    print()
    print('success')
```

(continues on next page)

(continued from previous page)

```
    print()
    print()

    """
    Query written data
    """
    query = f'from(bucket: "{bucket}") |> range(start: -1d) |> filter(fn: (r) => r._
↪measurement == "{kind}")'
    print(f'Querying from InfluxDB cloud: "{query}" ...')
    print()

    query_api = client.query_api()
    tables = query_api.query(query=query, org=org)

    for table in tables:
        for row in table.records:
            print(f'{row.values["_time"]}: host={row.values["host"]},device={row.
↪values["device"]} '
                  f'{row.values["_value"]} °C')

    print()
    print('success')

except Exception as e:
    print(e)
finally:
    client.close()
```

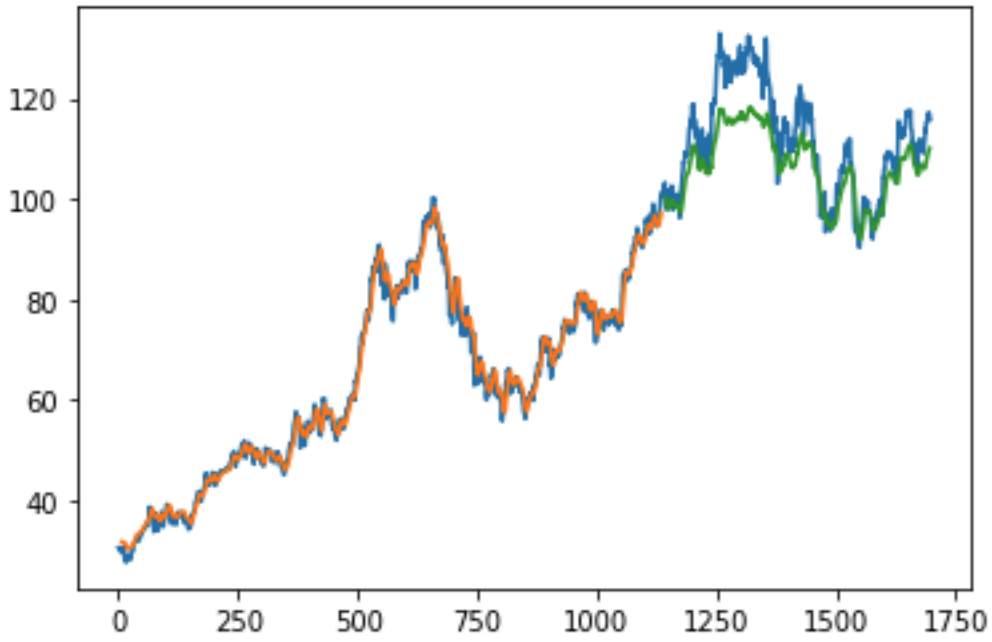
1.10.4 How to use Jupyter + Pandas + InfluxDB 2

The first example shows how to use client capabilities to predict stock price via Keras, TensorFlow, sklearn:

The example is taken from Kaggle.

- sources - stock-predictions.ipynb

Result:



The second example shows how to use client capabilities to realtime visualization via hvPlot, Streamz, RxPY:

- sources - realtime-stream.ipynb

1.10.5 Other examples

You can find all examples at GitHub: influxdb-client-python/examples.

CHAPTER 2

API Reference

2.1 InfluxDBClient

class influxdb_client.**InfluxDBClient**(*url*, *token*, *debug=None*, *timeout=10000*, *enable_gzip=False*, *org: str = None*, *default_tags: dict = None*, ***kwargs*)

InfluxDBClient is client for InfluxDB v2.

Initialize defaults.

Parameters

- **url** – InfluxDB server API url (ex. http://localhost:8086).
- **token** – auth token

- **debug** – enable verbose logging of http requests
- **timeout** – HTTP client timeout setting for a request specified in milliseconds. If one number provided, it will be total request timeout. It can also be a pair (tuple) of (connection, read) timeouts.
- **enable_gzip** – Enable Gzip compression for http requests. Currently only the "Write" and "Query" endpoints supports the Gzip compression.
- **org** – organization name (used as a default in query and write API)

Key bool verify_ssl Set this to false to skip verifying SSL certificate when calling API from https server.

Key str ssl_ca_cert Set this to customize the certificate file to verify the peer.

Key str proxy Set this to configure the http proxy to be used (ex. http://localhost:3128)

Key str proxy_headers A dictionary containing headers that will be sent to the proxy. Could be used for proxy authentication.

Key int connection_pool_maxsize Number of connections to save that can be reused by urllib3. Defaults to "multiprocessing.cpu_count() * 5".

Key urllib3.util.retry.Retry retries Set the default retry strategy that is used for all HTTP requests except batching writes. As a default there is no one retry strategy.

Key bool auth_basic Set this to true to enable basic authentication when talking to a InfluxDB 1.8.x that does not use auth-enabled but is protected by a reverse proxy with basic authentication. (defaults to false, don't set to true when talking to InfluxDB 2)

Key list[str] profilers list of enabled Flux profilers

authorizations_api() → influxdb_client.client.authorizations_api.AuthorizationsApi
Create the Authorizations API instance.

Returns authorizations api

buckets_api() → influxdb_client.client.bucket_api.BucketsApi
Create the Bucket API instance.

Returns buckets api

close()
Shutdown the client.

delete_api() → influxdb_client.client.delete_api.DeleteApi
Get the delete metrics API instance.

Returns delete api

classmethod from_config_file(*config_file: str = 'config.ini'*, *debug=None*, *enable_gzip=False*)
Configure client via configuration file. The configuration has to be under 'influx' section.

The supported formats:

- https://docs.python.org/3/library/configparser.html
- https://toml.io/en/

Configuration options:

- url
- org

- token
- timeout,
- verify_ssl
- ssl_ca_cert
- connection_pool_maxsize
- auth_basic
- profilers
- proxy

config.ini example:

```
[influx2]
url=http://localhost:8086
org=my-org
token=my-token
timeout=6000
connection_pool_maxsize=25
auth_basic=false
profilers=query,operator
proxy=http:proxy.domain.org:8080

[tags]
id = 132-987-655
customer = California Miner
data_center = ${env.data_center}
```

config.toml example:

```
[influx2]
    url = "http://localhost:8086"
    token = "my-token"
    org = "my-org"
    timeout = 6000
    connection_pool_maxsize = 25
    auth_basic = false
    profilers="query, operator"
    proxy = "http://proxy.domain.org:8080"

[tags]
    id = "132-987-655"
    customer = "California Miner"
    data_center = "${env.data_center}"
```

classmethod from_env_properties(*debug=None*, *enable_gzip=False*)

Configure client via environment properties.

Supported environment properties:

- INFLUXDB_V2_URL
- INFLUXDB_V2_ORG
- INFLUXDB_V2_TOKEN
- INFLUXDB_V2_TIMEOUT
- INFLUXDB_V2_VERIFY_SSL

- INFLUXDB_V2_SSL_CA_CERT
- INFLUXDB_V2_CONNECTION_POOL_MAXSIZE
- INFLUXDB_V2_AUTH_BASIC

health() → influxdb_client.domain.health_check.HealthCheck
Get the health of an instance.

Returns HealthCheck

labels_api() → influxdb_client.client.labels_api.LabelsApi
Create the Labels API instance.

Returns labels api

organizations_api() → influxdb_client.client.organizations_api.OrganizationsApi
Create the Organizations API instance.

Returns organizations api

ping() → bool
Return the the status of InfluxDB instance.

Returns The status of InfluxDB.

query_api(*query_options: influxdb_client.client.query_api.QueryOptions = <influxdb_client.client.query_api.QueryOptions object>*) → influxdb_client.client.query_api.QueryApi
Create a Query API instance.

Parameters **query_options** – optional query api configuration

Returns Query api instance

ready() → influxdb_client.domain.ready.Ready
Get The readiness of the InfluxDB 2.0.

Returns Ready

tasks_api() → influxdb_client.client.tasks_api.TasksApi
Create the Tasks API instance.

Returns tasks api

users_api() → influxdb_client.client.users_api.UsersApi
Create the Users API instance.

Returns users api

version() → str
Return the version of the connected InfluxDB Server.

Returns The version of InfluxDB.

write_api(*write_options=<influxdb_client.client.write_api.WriteOptions object>, point_settings=<influxdb_client.client.write_api.PointSettings object>, **kwargs*) → influxdb_client.client.write_api.WriteApi
Create a Write API instance.

Example:

```
from influxdb_client import InfluxDBClient
from influxdb_client.client.write_api import SYNCHRONOUS
```

(continues on next page)

(continued from previous page)

```
# Initialize SYNCHRONOUS instance of WriteApi
with InfluxDBClient(url="http://localhost:8086", token="my-token", org=
↪"my-org") as client:
    write_api = client.write_api(write_options=SYNCHRONOUS)
```

If you would like to use a **background batching**, you have to configure client like this:

```
from influxdb_client import InfluxDBClient

# Initialize background batching instance of WriteApi
with InfluxDBClient(url="http://localhost:8086", token="my-token", org="my-org
↪") as client:
    with client.write_api() as write_api:
        pass
```

There is also possibility to use callbacks to notify about state of background batches:

```
from influxdb_client import InfluxDBClient
from influxdb_client.client.exceptions import InfluxDBError


class BatchingCallback(object):

    def success(self, conf: (str, str, str), data: str):
        print(f"Written batch: {conf}, data: {data}")

    def error(self, conf: (str, str, str), data: str, exception:␣
↪InfluxDBError):
        print(f"Cannot write batch: {conf}, data: {data} due: {exception}")

    def retry(self, conf: (str, str, str), data: str, exception:␣
↪InfluxDBError):
        print(f"Retryable error occurs for batch: {conf}, data: {data} retry:
↪{exception}")


with InfluxDBClient(url="http://localhost:8086", token="my-token", org="my-org
↪") as client:
    callback = BatchingCallback()
    with client.write_api(success_callback=callback.success,
                          error_callback=callback.error,
                          retry_callback=callback.retry) as write_api:
        pass
```

Parameters

- **write_options** – Write API configuration
- **point_settings** – settings to store default tags

Key success_callback The callable `callback` to run after successfully writen a batch.

The callable must accept two arguments:

- *Tuple*: `(bucket, organization, precision)`
- *str*: written data

[batching mode]

Key error_callback The callable `callback` to run after unsuccessfully writen a batch.

The callable must accept three arguments:

- *Tuple*: `(bucket, organization, precision)`
- *str*: written data
- *Exception*: an occurred error

[batching mode]

Key retry_callback The callable `callback` to run after retryable error occurred.

The callable must accept three arguments:

- *Tuple*: `(bucket, organization, precision)`
- *str*: written data
- *Exception*: an retryable error

[batching mode]

Returns write api instance

2.2 QueryApi

class influxdb_client.**QueryApi**(*influxdb_client*, *query_options=<influxdb_client.client.query_api.QueryOptions object>*)

Implementation for '/api/v2/query' endpoint.

Initialize query client.

Parameters **influxdb_client** – influxdb client

query(*query: str*, *org=None*, *params: dict = None*) → List[influxdb_client.client.flux_table.FluxTable]

Execute synchronous Flux query and return result as a List['FluxTable'].

Parameters

- **query** – the Flux query
- **Organization org** (*str,*) – specifies the organization for executing the query; take the ID, Name or Organization; if it's not specified then is used default from client.org.
- **params** – bind parameters

Returns

query_csv(*query: str, org=None, dialect: influxdb_client.domain.dialect.Dialect = {'annotations': ['datatype', 'group', 'default'], 'comment_prefix': '#', 'date_time_format': 'RFC3339', 'delimiter': ',', 'header': True}, params: dict = None*)

Execute the Flux query and return results as a CSV iterator. Each iteration returns a row of the CSV file.

Parameters

- **query** – a Flux query
- **Organization org** (*str,*) – specifies the organization for executing the query; take the ID, Name or Organization; if it's not specified then is used default from client.org.
- **dialect** – csv dialect format

- **params** – bind parameters

Returns The returned object is an iterator. Each iteration returns a row of the CSV file (which can span multiple input lines).

query_data_frame(*query: str*, *org=None*, *data_frame_index: List[str] = None*, *params: dict = None*)

Execute synchronous Flux query and return Pandas DataFrame.

Note that if a query returns tables with differing schemas than the client generates a DataFrame for each of them.

Parameters

- **query** – the Flux query
- **Organization org** (*str,*) – specifies the organization for executing the query; take the ID, Name or Organization; if it's not specified then is used default from client.org.
- **data_frame_index** – the list of columns that are used as DataFrame index
- **params** – bind parameters

Returns

query_data_frame_stream(*query: str*, *org=None*, *data_frame_index: List[str] = None*, *params: dict = None*)

Execute synchronous Flux query and return stream of Pandas DataFrame as a Generator[‘pd.DataFrame’].

Note that if a query returns tables with differing schemas than the client generates a DataFrame for each of them.

Parameters

- **query** – the Flux query
- **Organization org** (*str,*) – specifies the organization for executing the query; take the ID, Name or Organization; if it's not specified then is used default from client.org.
- **data_frame_index** – the list of columns that are used as DataFrame index
- **params** – bind parameters

Returns

query_raw(*query: str, org=None, dialect={'annotations': ['datatype', 'group', 'default'], 'comment_prefix': '#', 'date_time_format': 'RFC3339', 'delimiter': ',', 'header': True}, params: dict = None*)

Execute synchronous Flux query and return result as raw unprocessed result as a str.

Parameters

- **query** – a Flux query
- **Organization org** (*str,*) – specifies the organization for executing the query; take the ID, Name or Organization; if it's not specified then is used default from client.org.
- **dialect** – csv dialect format
- **params** – bind parameters

Returns str

query_stream(*query: str*, *org=None*, *params: dict = None*) → Generator[[influxdb_client.client.flux_table.FluxRecord, Any], None]

Execute synchronous Flux query and return stream of FluxRecord as a Generator[‘FluxRecord’].

Parameters

- **query** – the Flux query
- **Organization org** (*str,*) – specifies the organization for executing the query; take the ID, Name or Organization; if it's not specified then is used default from client.org.
- **params** – bind parameters

Returns

2.3 WriteApi

class influxdb_client.**WriteApi**(*influxdb_client*, *write_options: influxdb_client.client.write_api.WriteOptions = <influxdb_client.client.write_api.WriteOptions object>*, *point_settings: influxdb_client.client.write_api.PointSettings = <influxdb_client.client.write_api.PointSettings object>*, ***kwargs*)

Implementation for '/api/v2/write' endpoint.

Example:

```
from influxdb_client import InfluxDBClient
from influxdb_client.client.write_api import SYNCHRONOUS


# Initialize SYNCHRONOUS instance of WriteApi
with InfluxDBClient(url="http://localhost:8086", token="my-token", org="my-org
↪") as client:
    write_api = client.write_api(write_options=SYNCHRONOUS)
```

Initialize defaults.

Parameters

- **influxdb_client** – with default settings (organization)
- **write_options** – write api configuration
- **point_settings** – settings to store default tags.

Key success_callback The callable `callback` to run after successfully writen a batch.

The callable must accept two arguments:

- *Tuple*: `(bucket, organization, precision)`
- *str*: written data

[batching mode]

Key error_callback The callable `callback` to run after unsuccessfully writen a batch.

The callable must accept three arguments:

- *Tuple*: `(bucket, organization, precision)`
- *str*: written data
- *Exception*: an occurred error

[batching mode]

Key retry_callback The callable `callback` to run after retryable error occurred.

The callable must accept three arguments:

- *Tuple*: `(bucket, organization, precision)`
- *str*: written data
- *Exception*: an retryable error

[batching mode]

close()
Flush data and dispose a batching buffer.

flush()
Flush data.

write(*bucket: str, org: str = None, record: Union[str, Iterable[str], influxdb_client.client.write.point.Point, Iterable[Point], dict, Iterable[dict], bytes, Iterable[bytes], rx.core.observable.observable.Observable, NamedTuple, Iterable[NamedTuple], dataclass, Iterable[dataclass]] = None, write_precision: influxdb_client.domain.write_precision.WritePrecision = 'ns', **kwargs*) → Any
Write time-series data into InfluxDB.

Parameters

- **bucket** (*str*) – specifies the destination bucket for writes (required)
- **Organization org** (*str,*) – specifies the destination organization for writes; take the ID, Name or Organization; if it's not specified then is used default from client.org.
- **write_precision** (WritePrecision) – specifies the precision for the unix timestamps within the body line-protocol. The precision specified on a Point has precedes and is use for write.
- **record** – Point, Line Protocol, Dictionary, NamedTuple, Data Classes, Pandas DataFrame or RxPY Observable to write

Key data_frame_measurement_name name of measurement for writing Pandas DataFrame - `DataFrame`

Key data_frame_tag_columns list of DataFrame columns which are tags, rest columns will be fields - `DataFrame`

Key record_measurement_key key of record with specified measurement - `dictionary`, `NamedTuple`, `dataclass`

Key record_measurement_name static measurement name - `dictionary`, `NamedTuple`, `dataclass`

Key record_time_key key of record with specified timestamp - `dictionary`, `NamedTuple`, `dataclass`

Key record_tag_keys list of record keys to use as a tag - `dictionary`, `NamedTuple`, `dataclass`

Key record_field_keys list of record keys to use as a field - `dictionary`, `NamedTuple`, `dataclass`

Example:

```
# Record as Line Protocol
write_api.write("my-bucket", "my-org", "h2o_feet,location=us-west␣
↪level=125i 1")
```

(continues on next page)

(continued from previous page)

```
# Record as Dictionary
dictionary = {
    "measurement": "h2o_feet",
    "tags": {"location": "us-west"},
    "fields": {"level": 125},
    "time": 1
}
write_api.write("my-bucket", "my-org", dictionary)

# Record as Point
from influxdb_client import Point
point = Point("h2o_feet").tag("location", "us-west").field("level", 125).
↪time(1)
write_api.write("my-bucket", "my-org", point)
```

DataFrame: The index of Pandas DataFrame is used as a `timestamp` for written data. The index should be PeriodIndex or its must be transformable to `datetime` by pandas.to_datetime.

If you would like to transform a column to `PeriodIndex`, you can use something like:

```
import pandas as pd

# DataFrame
data_frame = ...
# Set column as Index
data_frame.set_index('column_name', inplace=True)
# Transform index to PeriodIndex
data_frame.index = pd.to_datetime(data_frame.index, unit='s')
```

class influxdb_client.client.write.point.**Point**(*measurement_name*)

Point defines the values that will be written to the database.

Ref: http://bit.ly/influxdata-point

Initialize defaults.

field(*field*, *value*)

Add field with key and value.

static from_dict(*dictionary: dict*, *write_precision: influxdb_client.domain.write_precision.WritePrecision = 'ns'*, ***kwargs*)

Initialize point from 'dict' structure.

The expected dict structure is:

- measurement
- tags
- fields
- time

Example:

```
# Use default dictionary structure
dict_structure = {
    "measurement": "h2o_feet",
    "tags": {"location": "coyote_creek"},
```

(continues on next page)

(continued from previous page)

```
    "fields": {"water_level": 1.0},
    "time": 1
}
point = Point.from_dict(dict_structure, WritePrecision.NS)
```

Example:

```
# Use custom dictionary structure
dictionary = {
    "name": "sensor_pt859",
    "location": "warehouse_125",
    "version": "2021.06.05.5874",
    "pressure": 125,
    "temperature": 10,
    "created": 1632208639,
}
point = Point.from_dict(dictionary,
                        write_precision=WritePrecision.S,
                        record_measurement_key="name",
                        record_time_key="created",
                        record_tag_keys=["location", "version"],
                        record_field_keys=["pressure", "temperature"])
```

Parameters

- **dictionary** – dictionary for serialize into data Point
- **write_precision** – sets the precision for the supplied time values

Key record_measurement_key key of dictionary with specified measurement

Key record_measurement_name static measurement name for data Point

Key record_time_key key of dictionary with specified timestamp

Key record_tag_keys list of dictionary keys to use as a tag

Key record_field_keys list of dictionary keys to use as a field

Returns new data point

static measurement(*measurement*)
Create a new Point with specified measurement name.

tag(*key*, *value*)
Add tag with key and value.

time(*time*, *write_precision='ns'*)
Specify timestamp for DataPoint with declared precision.

If time doesn't have specified timezone we assume that timezone is UTC.

Examples:: Point.measurement("h2o").field("val", 1).time("2009-11-10T23:00:00.123456Z") Point.measurement("h2o").field("val", 1).time(1257894000123456000) Point.measurement("h2o").field("val", 1).time(datetime(2009, 11, 10, 23, 0, 0, 123456)) Point.measurement("h2o").field("val", 1).time(1257894000123456000, write_precision=WritePrecision.NS)

Parameters

- **time** – the timestamp for your data
- **write_precision** – sets the precision for the supplied time values

Returns this point

to_line_protocol()
Create LineProtocol.

write_precision
Get precision.

class influxdb_client.domain.write_precision.**WritePrecision**
NOTE: This class is auto generated by OpenAPI Generator.

Ref: https://openapi-generator.tech

Do not edit the class manually.

WritePrecision - a model defined in OpenAPI.

NS = 'ns'

Attributes:

openapi_types (dict): The key is attribute name and the value is attribute type.

attribute_map (dict): The key is attribute name and the value is json key in definition.

to_dict()
Return the model properties as a dict.

to_str()
Return the string representation of the model.

2.4 BucketsApi

class influxdb_client.**BucketsApi**(*influxdb_client*)
Implementation for '/api/v2/buckets' endpoint.

Initialize defaults.

create_bucket(*bucket=None*, *bucket_name=None*, *org_id=None*, *retention_rules=None*, *description=None*, *org=None*) → influxdb_client.domain.bucket.Bucket
Create a bucket.

Parameters

- **bucket** (Bucket) – bucket to create
- **bucket_name** – bucket name
- **description** – bucket description
- **org_id** – org_id
- **bucket_name** – bucket name
- **retention_rules** – retention rules array or single BucketRetentionRules
- **Organization org**(*str,*) – specifies the organization for create the bucket; take the ID, Name or Organization; if it's not specified then is used default from client.org.

Returns Bucket If the method is called asynchronously, returns the request thread.

delete_bucket(*bucket*)
Delete a bucket.

Parameters **bucket** – bucket id or Bucket

Returns Bucket

find_bucket_by_id(*id*)
Find bucket by ID.

Parameters **id** –

Returns

find_bucket_by_name(*bucket_name*)
Find bucket by name.

Parameters **bucket_name** – bucket name

Returns Bucket

find_buckets(***kwargs*)
List buckets.

Key int offset Offset for pagination

Key int limit Limit for pagination

Key str after The last resource ID from which to seek from (but not including). This is to be used instead of *offset*.

Key str org The organization name.

Key str org_id The organization ID.

Key str name Only returns buckets with a specific name.

Returns Buckets

update_bucket(*bucket: influxdb_client.domain.bucket.Bucket*) → influxdb_client.domain.bucket.Bucket
Update a bucket.

Parameters **bucket** – Bucket update to apply (required)

Returns Bucket

class influxdb_client.domain.**Bucket**(*links=None*, *id=None*, *type='user'*, *name=None*, *description=None*, *org_id=None*, *rp=None*, *schema_type=None*, *created_at=None*, *updated_at=None*, *retention_rules=None*, *labels=None*)
NOTE: This class is auto generated by OpenAPI Generator.

Ref: https://openapi-generator.tech

Do not edit the class manually.

Bucket - a model defined in OpenAPI.

created_at
Get the created_at of this Bucket.

Returns The created_at of this Bucket.

Return type datetime

description
Get the description of this Bucket.

Returns The description of this Bucket.

Return type str

id
Get the id of this Bucket.

Returns The id of this Bucket.

Return type str

labels
Get the labels of this Bucket.

Returns The labels of this Bucket.

Return type list[Label]

links
Get the links of this Bucket.

Returns The links of this Bucket.

Return type BucketLinks

name
Get the name of this Bucket.

Returns The name of this Bucket.

Return type str

org_id
Get the org_id of this Bucket.

Returns The org_id of this Bucket.

Return type str

retention_rules
Get the retention_rules of this Bucket.

Rules to expire or retain data. No rules means data never expires.

Returns The retention_rules of this Bucket.

Return type list[BucketRetentionRules]

rp
Get the rp of this Bucket.

Returns The rp of this Bucket.

Return type str

schema_type
Get the schema_type of this Bucket.

Returns The schema_type of this Bucket.

Return type SchemaType

to_dict()
Return the model properties as a dict.

to_str()
Return the string representation of the model.

type
Get the type of this Bucket.

Returns The type of this Bucket.

Return type str

updated_at
Get the updated_at of this Bucket.

Returns The updated_at of this Bucket.

Return type datetime

2.5 LabelsApi

class influxdb_client.**LabelsApi**(*influxdb_client*)
Implementation for '/api/v2/labels' endpoint.

Initialize defaults.

clone_label(*cloned_name: str*, *label: influxdb_client.domain.label.Label*) → influxdb_client.domain.label.Label
Create the new instance of the label as a copy existing label.

Parameters

- **cloned_name** – new label name
- **label** – existing label

Returns clonned Label

create_label(*name: str*, *org_id: str*, *properties: Dict[str, str] = None*) → influxdb_client.domain.label.Label
Create a new label.

Parameters

- **name** – label name
- **org_id** – organization id
- **properties** – optional label properties

Returns created label

delete_label(*label: Union[str, influxdb_client.domain.label.Label]*)
Delete the label.

Parameters **label** – label id or Label

find_label_by_id(*label_id: str*)
Retrieve the label by id.

Parameters **label_id** –

Returns Label

find_label_by_org(*org_id*) → List[influxdb_client.domain.label.Label]
Get the list of all labels for given organization.

Parameters **org_id** – organization id

Returns list of labels

find_labels(**kwargs) → List[influxdb_client.domain.label.Label]
Get all available labels.

Key str org_id The organization ID.

Returns labels

update_label(*label: influxdb_client.domain.label.Label*)
Update an existing label name and properties.

Parameters **label** – label

Returns the updated label

2.6 OrganizationsApi

class influxdb_client.**OrganizationsApi**(*influxdb_client*)
Implementation for '/api/v2/orgs' endpoint.

Initialize defaults.

create_organization(*name: str = None*, *organization: influxdb_client.domain.organization.Organization = None*) → influxdb_client.domain.organization.Organization
Create an organization.

delete_organization(*org_id: str*)
Delete an organization.

find_organization(*org_id*)
Retrieve an organization.

find_organizations(**kwargs)
List all organizations.

Key int offset Offset for pagination

Key int limit Limit for pagination

Key bool descending

Key str org Filter organizations to a specific organization name.

Key str org_id Filter organizations to a specific organization ID.

Key str user_id Filter organizations to a specific user ID.

me()
Return the current authenticated user.

update_organization(*organization: influxdb_client.domain.organization.Organization*) → influxdb_client.domain.organization.Organization
Update an organization.

Parameters **organization** – Organization update to apply (required)

Returns Organization

class influxdb_client.domain.**Organization**(*links=None*, *id=None*, *name=None*, *description=None*, *created_at=None*, *updated_at=None*, *status='active'*)
NOTE: This class is auto generated by OpenAPI Generator.

Ref: https://openapi-generator.tech

Do not edit the class manually.

Organization - a model defined in OpenAPI.

created_at
Get the created_at of this Organization.

Returns The created_at of this Organization.

Return type datetime

description
Get the description of this Organization.

Returns The description of this Organization.

Return type str

id
Get the id of this Organization.

Returns The id of this Organization.

Return type str

links
Get the links of this Organization.

Returns The links of this Organization.

Return type OrganizationLinks

name
Get the name of this Organization.

Returns The name of this Organization.

Return type str

status
Get the status of this Organization.

If inactive the organization is inactive.

Returns The status of this Organization.

Return type str

to_dict()
Return the model properties as a dict.

to_str()
Return the string representation of the model.

updated_at
Get the updated_at of this Organization.

Returns The updated_at of this Organization.

Return type datetime

2.7 UsersApi

class influxdb_client.**UsersApi**(*influxdb_client*)
Implementation for '/api/v2/users' endpoint.

Initialize defaults.

create_user(*name: str*) → influxdb_client.domain.user.User
Create a user.

delete_user(*user: Union[str, influxdb_client.domain.user.User, influxdb_client.domain.user_response.UserResponse]*) → None
Delete a user.

Parameters **user** – user id or User

Returns User

find_users(***kwargs*) → influxdb_client.domain.users.Users
List all users.

Key int offset Offset for pagination

Key int limit Limit for pagination

Key str after The last resource ID from which to seek from (but not including). This is to be used instead of *offset*.

Key str name The user name.

Key str id The user ID.

Returns Buckets

me() → influxdb_client.domain.user.User
Return the current authenticated user.

update_user(*user: influxdb_client.domain.user.User*) → influxdb_client.domain.user_response.UserResponse
Update a user.

Parameters **user** – User update to apply (required)

Returns User

class influxdb_client.domain.**User**(*id=None*, *oauth_id=None*, *name=None*, *status='active'*)
NOTE: This class is auto generated by OpenAPI Generator.

Ref: https://openapi-generator.tech

Do not edit the class manually.

User - a model defined in OpenAPI.

id
Get the id of this User.

Returns The id of this User.

Return type str

name
Get the name of this User.

Returns The name of this User.

Return type str

oauth_id
Get the oauth_id of this User.

Returns The oauth_id of this User.

Return type str

status
Get the status of this User.

If inactive the user is inactive.

Returns The status of this User.

Return type str

to_dict()
Return the model properties as a dict.

to_str()
Return the string representation of the model.

2.8 TasksApi

class influxdb_client.**TasksApi**(*influxdb_client*)
Implementation for '/api/v2/tasks' endpoint.

Initialize defaults.

add_label(*label_id: str*, *task_id: str*) → influxdb_client.domain.label_response.LabelResponse
Add a label to a task.

add_member(*member_id*, *task_id*)
Add a member to a task.

add_owner(*owner_id*, *task_id*)
Add an owner to a task.

cancel_run(*task_id: str*, *run_id: str*)
Cancel a currently running run.

Parameters

- **task_id** –
- **run_id** –

clone_task(*task: influxdb_client.domain.task.Task*) → influxdb_client.domain.task.Task
Clone a task.

create_task(*task: influxdb_client.domain.task.Task = None*, *task_create_request: influxdb_client.domain.task_create_request.TaskCreateRequest = None*) → influxdb_client.domain.task.Task
Create a new task.

create_task_cron(*name: str*, *flux: str*, *cron: str*, *org_id: str*) → influxdb_client.domain.task.Task
Create a new task with cron repetition schedule.

create_task_every(*name*, *flux*, *every*, *organization*) → influxdb_client.domain.task.Task
Create a new task with every repetition schedule.

delete_label(*label_id: str*, *task_id: str*)
Delete a label from a task.

delete_member(*member_id*, *task_id*)
Remove a member from a task.

delete_owner(*owner_id*, *task_id*)
Remove an owner from a task.

delete_task(*task_id: str*)
Delete a task.

find_task_by_id(*task_id*) → influxdb_client.domain.task.Task
Retrieve a task.

find_tasks(***kwargs*)
List all tasks.

Key str name only returns tasks with the specified name

Key str after returns tasks after specified ID

Key str user filter tasks to a specific user ID

Key str org filter tasks to a specific organization name

Key str org_id filter tasks to a specific organization ID

Key int limit the number of tasks to return

Returns Tasks

find_tasks_by_user(*task_user_id*)
List all tasks by user.

get_labels(*task_id*)
List all labels for a task.

get_logs(*task_id: str*) → List[influxdb_client.domain.log_event.LogEvent]
Retrieve all logs for a task.

Parameters **task_id** – task id

get_members(*task_id: str*)
List all task members.

get_owners(*task_id*)
List all owners of a task.

get_run(*task_id: str*, *run_id: str*) → influxdb_client.domain.run.Run
Get run record for specific task and run id.

Parameters

- **task_id** – task id
- **run_id** – run id

Returns Run for specified task and run id

get_run_logs(*task_id: str*, *run_id: str*) → List[influxdb_client.domain.log_event.LogEvent]
Retrieve all logs for a run.

get_runs(*task_id*, ***kwargs*) → List[influxdb_client.domain.run.Run]
Retrieve list of run records for a task.

Parameters

- **task_id** – task id
- **after** (*str*) – returns runs after specified ID
- **limit** (*int*) – the number of runs to return
- **after_time** (*datetime*) – filter runs to those scheduled after this time, RFC3339
- **before_time** (*datetime*) – filter runs to those scheduled before this time, RFC3339

retry_run(*task_id: str*, *run_id: str*)
Retry a task run.

Parameters

- **task_id** – task id
- **run_id** – run id

run_manually(*task_id: str*, *scheduled_for: <module 'datetime' from '/home/docs/.pyenv/versions/3.6.12/lib/python3.6/datetime.py'> = None*)
Manually start a run of the task now overriding the current schedule.

Parameters

- **task_id** –
- **scheduled_for** – planned execution

update_task(*task: influxdb_client.domain.task.Task*) → influxdb_client.domain.task.Task
Update a task.

update_task_request(*task_id*, *task_update_request: influxdb_client.domain.task_update_request.TaskUpdateRequest*) → influxdb_client.domain.task.Task
Update a task.

class influxdb_client.domain.**Task**(*id=None*, *type=None*, *org_id=None*, *org=None*, *name=None*, *owner_id=None*, *description=None*, *status=None*, *labels=None*, *authorization_id=None*, *flux=None*, *every=None*, *cron=None*, *offset=None*, *latest_completed=None*, *last_run_status=None*, *last_run_error=None*, *created_at=None*, *updated_at=None*, *links=None*)
NOTE: This class is auto generated by OpenAPI Generator.

Ref: https://openapi-generator.tech

Do not edit the class manually.

Task - a model defined in OpenAPI.

authorization_id
Get the authorization_id of this Task.

The ID of the authorization used when this task communicates with the query engine.

Returns The authorization_id of this Task.

Return type str

created_at
Get the created_at of this Task.

Returns The created_at of this Task.

Return type datetime

cron
Get the cron of this Task.

A task repetition schedule in the form '* * * * * *'; parsed from Flux.

Returns The cron of this Task.

Return type str

description
Get the description of this Task.

An optional description of the task.

Returns The description of this Task.

Return type str

every
Get the every of this Task.

A simple task repetition schedule; parsed from Flux.

Returns The every of this Task.

Return type str

flux
Get the flux of this Task.

The Flux script to run for this task.

Returns The flux of this Task.

Return type str

id
Get the id of this Task.

Returns The id of this Task.

Return type str

labels
Get the labels of this Task.

Returns The labels of this Task.

Return type list[Label]

last_run_error
Get the last_run_error of this Task.

Returns The last_run_error of this Task.

Return type str

last_run_status
Get the last_run_status of this Task.

Returns The last_run_status of this Task.

Return type str

latest_completed
Get the latest_completed of this Task.

Timestamp of latest scheduled, completed run, RFC3339.

Returns The latest_completed of this Task.

Return type datetime

links
Get the links of this Task.

Returns The links of this Task.

Return type TaskLinks

name
Get the name of this Task.

The name of the task.

Returns The name of this Task.

Return type str

offset
Get the offset of this Task.

Duration to delay after the schedule, before executing the task; parsed from flux, if set to zero it will remove this option and use 0 as the default.

Returns The offset of this Task.

Return type str

org
Get the org of this Task.

The name of the organization that owns this Task.

Returns The org of this Task.

Return type str

org_id
Get the org_id of this Task.

The ID of the organization that owns this Task.

Returns The org_id of this Task.

Return type str

owner_id
Get the owner_id of this Task.

The ID of the user who owns this Task.

Returns The owner_id of this Task.

Return type str

status
Get the status of this Task.

Returns The status of this Task.

Return type TaskStatusType

to_dict()
Return the model properties as a dict.

to_str()
Return the string representation of the model.

type
Get the type of this Task.

The type of task, this can be used for filtering tasks on list actions.

Returns The type of this Task.

Return type str

updated_at
Get the updated_at of this Task.

Returns The updated_at of this Task.

Return type datetime

2.9 DeleteApi

class influxdb_client.**DeleteApi**(*influxdb_client*)
Implementation for '/api/v2/delete' endpoint.

Initialize defaults.

delete(*start: datetime.datetime*, *stop: object*, *predicate: object*, *bucket: str*, *org: str*) → None
Delete Time series data from InfluxDB.

Parameters

- **start** – start time
- **stop** – stop time
- **predicate** – predicate
- **bucket** – bucket id or name from which data will be deleted
- **org** – organization id or name

Returns

class influxdb_client.domain.**DeletePredicateRequest**(*start=None*, *stop=None*, *predicate=None*)
NOTE: This class is auto generated by OpenAPI Generator.

Ref: https://openapi-generator.tech

Do not edit the class manually.

DeletePredicateRequest - a model defined in OpenAPI.

predicate
Get the predicate of this DeletePredicateRequest.

InfluxQL-like delete statement

Returns The predicate of this DeletePredicateRequest.

Return type str

start
Get the start of this DeletePredicateRequest.

RFC3339Nano

Returns The start of this DeletePredicateRequest.

Return type datetime

stop
Get the stop of this DeletePredicateRequest.

RFC3339Nano

Returns The stop of this DeletePredicateRequest.

Return type datetime

to_dict()
Return the model properties as a dict.

to_str()
Return the string representation of the model.

2.10 Helpers

class influxdb_client.client.util.date_utils.**DateHelper**(*timezone: datetime.tzinfo = <UTC>*)
DateHelper to groups different implementations of date operations.

Initialize defaults.

Parameters **timezone** – Default timezone used for serialization "datetime" without "tzinfo". Default value is "UTC".

parse_date(*date_string: str*)
Parse string into Date or Timestamp.

Returns Returns a `datetime.datetime` object or compliant implementation like `class 'pandas._libs.tslibs.timestamps.Timestamp`

to_nanoseconds(*delta*)
Get number of nanoseconds in timedelta.

Solution comes from v1 client. Thx. https://github.com/influxdata/influxdb-python/pull/811

to_utc(*value: <module 'datetime' from '/home/docs/.pyenv/versions/3.6.12/lib/python3.6/datetime.py'>*)
Convert datetime to UTC timezone.

Parameters **value** – datetime

Returns datetime in UTC

class influxdb_client.client.util.multiprocessing_helper.**MultiprocessingWriter**(***kwargs*)
The Helper class to write data into InfluxDB in independent OS process.

Example:

```
from influxdb_client import WriteOptions
from influxdb_client.client.util.multiprocessing_helper import␣
↪MultiprocessingWriter


def main():
    writer = MultiprocessingWriter(url="http://localhost:8086", token="my-
↪token", org="my-org",
                                   write_options=WriteOptions(batch_size=100))
    writer.start()

    for x in range(1, 1000):
        writer.write(bucket="my-bucket", record=f"mem,tag=a value={x}i {x}")

    writer.__del__()
```

(continues on next page)

(continued from previous page)

```
if __name__ == '__main__':
    main()
```

How to use with context_manager:

```
from influxdb_client import WriteOptions
from influxdb_client.client.util.multiprocessing_helper import␣
↪MultiprocessingWriter


def main():
    with MultiprocessingWriter(url="http://localhost:8086", token="my-token",␣
↪org="my-org",
                               write_options=WriteOptions(batch_size=100)) as␣
↪writer:
        for x in range(1, 1000):
            writer.write(bucket="my-bucket", record=f"mem,tag=a value={x}i {x}
↪")


if __name__ == '__main__':
    main()
```

How to handle batch events:

```
from influxdb_client import WriteOptions
from influxdb_client.client.exceptions import InfluxDBError
from influxdb_client.client.util.multiprocessing_helper import␣
↪MultiprocessingWriter


class BatchingCallback(object):

    def success(self, conf: (str, str, str), data: str):
        print(f"Written batch: {conf}, data: {data}")

    def error(self, conf: (str, str, str), data: str, exception:␣
↪InfluxDBError):
        print(f"Cannot write batch: {conf}, data: {data} due: {exception}")

    def retry(self, conf: (str, str, str), data: str, exception:␣
↪InfluxDBError):
        print(f"Retryable error occurs for batch: {conf}, data: {data} retry:
↪{exception}")


def main():
    callback = BatchingCallback()
    with MultiprocessingWriter(url="http://localhost:8086", token="my-token",␣
↪org="my-org",
                               success_callback=callback.success,
                               error_callback=callback.error,
                               retry_callback=callback.retry) as writer:

        for x in range(1, 1000):
            writer.write(bucket="my-bucket", record=f"mem,tag=a value={x}i {x}
↪")
```

(continues on next page)

(continued from previous page)

```
if __name__ == '__main__':
    main()
```

Initialize defaults.

For more information how to initialize the writer see the examples above.

Parameters **kwargs** – arguments are passed into `__init__` function of `InfluxDBClient` and `write_api`.

run()
Initialize `InfluxDBClient` and waits for data to writes into InfluxDB.

start() → None
Start independent process for writing data into InfluxDB.

terminate() → None
Cleanup resources in independent process.

This function **cannot be used** to terminate the `MultiprocessingWriter`. If you want to finish your writes please call: `__del__`.

write(***kwargs*) → None
Append time-series data into underlying queue.

For more information how to pass arguments see the examples above.

Parameters **kwargs** – arguments are passed into `write` function of `WriteApi`

Returns None

CHAPTER 3

Migration Guide

This guide is meant to help you migrate your Python code from influxdb-python to `influxdb-client-python` by providing code examples that cover common usages.

If there is something missing, please feel free to create a new request for a guide enhancement.

3.1 Before You Start

Please take a moment to review the following client docs:

- User Guide, README.rst
- Examples
- API Reference
- CHANGELOG.md

3.2 Content

- *Querying*

3.3 Initializing Client

influxdb-python

```
from influxdb import InfluxDBClient

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

influxdb-client-python

```
from influxdb_client import InfluxDBClient

with InfluxDBClient(url='http://localhost:8086', token='my-token', org='my-org') as␣
↪client:
    pass
```

3.4 Creating Database/Bucket

influxdb-python

```
from influxdb import InfluxDBClient

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

dbname = 'example'
client.create_database(dbname)
client.create_retention_policy('awesome_policy', '60m', 3, database=dbname,␣
↪default=True)
```

influxdb-client-python

```
from influxdb_client import InfluxDBClient, BucketRetentionRules

org = 'my-org'

with InfluxDBClient(url='http://localhost:8086', token='my-token', org=org) as client:
    buckets_api = client.buckets_api()

    # Create Bucket with retention policy set to 3600 seconds and name "bucket-by-
↪python"
    retention_rules = BucketRetentionRules(type="expire", every_seconds=3600)
    created_bucket = buckets_api.create_bucket(bucket_name="bucket-by-python",
                                               retention_rules=retention_rules,
                                               org=org)
```

3.5 Dropping Database/Bucket

influxdb-python

```
from influxdb import InfluxDBClient

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

dbname = 'example'
client.drop_database(dbname)
```

influxdb-client-python

```
from influxdb_client import InfluxDBClient

with InfluxDBClient(url='http://localhost:8086', token='my-token', org='my-org') as␣
↪client:
    buckets_api = client.buckets_api()

    bucket = buckets_api.find_bucket_by_name("my-bucket")
    buckets_api.delete_bucket(bucket)
```

3.6 Writing LineProtocol

influxdb-python

```
from influxdb import InfluxDBClient

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

client.write('h2o_feet,location=coyote_creek water_level=1.0 1', protocol='line')
```

influxdb-client-python

```
from influxdb_client import InfluxDBClient
from influxdb_client.client.write_api import SYNCHRONOUS

with InfluxDBClient(url='http://localhost:8086', token='my-token', org='my-org') as␣
↪client:
    write_api = client.write_api(write_options=SYNCHRONOUS)

    write_api.write(bucket='my-bucket', record='h2o_feet,location=coyote_creek water_
↪level=1.0 1')
```

3.7 Writing Dictionary-style object

influxdb-python

```
from influxdb import InfluxDBClient

record = [
        {
            "measurement": "cpu_load_short",
            "tags": {
```

(continues on next page)

(continued from previous page)

```
                "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='127.0.0.1', port=8086, username='root', password='root',
↪ database='dbname')

client.write_points(record)
```

influxdb-client-python

```
from influxdb_client import InfluxDBClient
from influxdb_client.client.write_api import SYNCHRONOUS

with InfluxDBClient(url='http://localhost:8086', token='my-token', org='my-org') as␣
↪client:
    write_api = client.write_api(write_options=SYNCHRONOUS)

    record = [
        {
            "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
            }
        }
    ]

    write_api.write(bucket='my-bucket', record=record)
```

3.8 Writing Structured Data

influxdb-python

```
from influxdb import InfluxDBClient
from influxdb import SeriesHelper

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

(continues on next page)

(continued from previous page)

```
class MySeriesHelper(SeriesHelper):
    class Meta:
        client = my_client
        series_name = 'events.stats.{server_name}'
        fields = ['some_stat', 'other_stat']
        tags = ['server_name']
        bulk_size = 5
        autocommit = True


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.commit()
```

The `influxdb-client-python` doesn't have an equivalent implementation for `MySeriesHelper`, but there is an option to use Python Data Classes way:

influxdb-client-python

```
from dataclasses import dataclass

from influxdb_client import InfluxDBClient
from influxdb_client.client.write_api import SYNCHRONOUS


@dataclass
class Car:
    """
    DataClass structure - Car
    """
    engine: str
    type: str
    speed: float


with InfluxDBClient(url='http://localhost:8086', token='my-token', org='my-org') as␣
↪client:
    write_api = client.write_api(write_options=SYNCHRONOUS)

    car = Car('12V-BT', 'sport-cars', 125.25)

    write_api.write(bucket="my-bucket",
                    record=car,
                    record_measurement_name="performance",
                    record_tag_keys=["engine", "type"],
                    record_field_keys=["speed"])
```

3.9 Writing Pandas DataFrame

influxdb-python

```
import pandas as pd

from influxdb import InfluxDBClient

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

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

client.write_points(df, 'demo', protocol='line')
```

influxdb-client-python

```
import pandas as pd

from influxdb_client import InfluxDBClient
from influxdb_client.client.write_api import SYNCHRONOUS

with InfluxDBClient(url='http://localhost:8086', token='my-token', org='my-org') as␣
↪client:
    write_api = client.write_api(write_options=SYNCHRONOUS)

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

    write_api.write(bucket='my-bucket', record=df, data_frame_measurement_name='demo')
```

3.10 Querying

influxdb-python

```
from influxdb import InfluxDBClient

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

points = client.query('SELECT * from cpu').get_points()
for point in points:
    print(point)
```

influxdb-client-python

```
from influxdb_client import InfluxDBClient

with InfluxDBClient(url='http://localhost:8086', token='my-token', org='my-org',␣
↪debug=True) as client:
    query = '''from(bucket: "my-bucket")
  |> range(start: -10000d)
  |> filter(fn: (r) => r["_measurement"] == "cpu")
  |> pivot(rowKey:["_time"], columnKey: ["_field"], valueColumn: "_value")
'''
```

(continues on next page)

(continued from previous page)

```
    tables = client.query_api().query(query)
    for record in [record for table in tables for record in table.records]:
        print(record.values)
```

If you would like to omit boilerplate columns such as `_result`, `_table`, `_start`, ... you can filter the record values by following expression:

```
print({k: v for k, v in record.values.items() if k not in ['result', 'table', '_start
↪', '_stop', '_measurement']})
```

For more info see Flux Response Format.

This repository contains the Python client library for the InfluxDB 2.0.

Note: Use this client library with InfluxDB 2.x and InfluxDB 1.8+. For connecting to InfluxDB 1.7 or earlier instances, use the influxdb-python **client library.** The API of the **influxdb-client-python** is not the backwards-compatible with the old one - **influxdb-python**.

CHAPTER 4

Documentation

This section contains links to the client library documentation.

- Product documentation, *Getting Started*
- Examples
- API Reference
- Changelog

CHAPTER 5

InfluxDB 2.0 client features

- **Querying data**
  - using the Flux language
  - into csv, raw data, flux_table structure, Pandas DataFrame
  - *How to queries*
- **Writing data using**
  - Line Protocol
  - Data Point
  - RxPY Observable
  - Pandas DataFrame
  - *How to writes*
- **InfluxDB 2.0 API client for management**
  - the client is generated from the swagger by using the openapi-generator
  - organizations & users management
  - buckets management
  - tasks management
  - authorizations
  - health check
  - ...
- **'InfluxDB 1.8 API compatibility'_**
- **Examples**
  - **'Connect to InfluxDB Cloud'_**
  - **'How to efficiently import large dataset'_**

  - ‘Efficiency write data from IOT sensor‘_
  - ‘How to use Jupyter + Pandas + InfluxDB 2‘_
- ‘Advanced Usage‘_
  - ‘Gzip support‘_
  - ‘Proxy configuration‘_
  - ‘Nanosecond precision‘_
  - ‘Delete data‘_
  - ‘Handling Errors‘_

CHAPTER 6

Installation

InfluxDB python library uses RxPY - The Reactive Extensions for Python (RxPY).

Python 3.6 or later is required.

Note: It is recommended to use `ciso8601` with client for parsing dates. `ciso8601` is much faster than built-in Python datetime. Since it's written as a `C` module the best way is build it from sources:

Windows:

You have to install Visual C++ Build Tools 2015 to build `ciso8601` by `pip`.

conda:

Install from sources: `conda install -c conda-forge/label/cf202003 ciso8601`.

6.1 pip install

The python package is hosted on PyPI, you can install latest version directly:

```
pip install 'influxdb-client[ciso]'
```

Then import the package:

```
import influxdb_client
```

6.2 Setuptools

Install via Setuptools.

```
python setup.py install --user
```

(or `sudo python setup.py install` to install the package for all users)

CHAPTER 7

Getting Started

Please follow the *Installation* and then run the following:

```
from influxdb_client import InfluxDBClient, Point
from influxdb_client.client.write_api import SYNCHRONOUS

bucket = "my-bucket"

client = InfluxDBClient(url="http://localhost:8086", token="my-token", org="my-org")

write_api = client.write_api(write_options=SYNCHRONOUS)
query_api = client.query_api()

p = Point("my_measurement").tag("location", "Prague").field("temperature", 25.3)

write_api.write(bucket=bucket, record=p)

## using Table structure
tables = query_api.query('from(bucket:"my-bucket") |> range(start: -10m)')

for table in tables:
    print(table)
    for row in table.records:
        print (row.values)


## using csv library
csv_result = query_api.query_csv('from(bucket:"my-bucket") |> range(start: -10m)')
val_count = 0
for row in csv_result:
    for cell in row:
        val_count += 1
```

CHAPTER 8

Client configuration

8.1 Via File

A client can be configured via `*.ini` file in segment `influx2`.

The following options are supported:

- `url` - the url to connect to InfluxDB
- `org` - default destination organization for writes and queries
- `token` - the token to use for the authorization
- `timeout` - socket timeout in ms (default value is 10000)
- `verify_ssl` - set this to false to skip verifying SSL certificate when calling API from https server
- `ssl_ca_cert` - set this to customize the certificate file to verify the peer
- `connection_pool_maxsize` - set the number of connections to save that can be reused by urllib3
- `auth_basic` - enable http basic authentication when talking to a InfluxDB 1.8.x without authentication but is accessed via reverse proxy with basic authentication (defaults to false)
- `profilers` - set the list of enabled Flux profilers

```
self.client = InfluxDBClient.from_config_file("config.ini")
```

8.2 Via Environment Properties

A client can be configured via environment properties.

Supported properties are:

- `INFLUXDB_V2_URL` - the url to connect to InfluxDB
- `INFLUXDB_V2_ORG` - default destination organization for writes and queries

- INFLUXDB_V2_TOKEN - the token to use for the authorization
- INFLUXDB_V2_TIMEOUT - socket timeout in ms (default value is 10000)
- INFLUXDB_V2_VERIFY_SSL - set this to false to skip verifying SSL certificate when calling API from https server
- INFLUXDB_V2_SSL_CA_CERT - set this to customize the certificate file to verify the peer
- INFLUXDB_V2_CONNECTION_POOL_MAXSIZE - set the number of connections to save that can be reused by urllib3
- INFLUXDB_V2_AUTH_BASIC - enable http basic authentication when talking to a InfluxDB 1.8.x without authentication but is accessed via reverse proxy with basic authentication (defaults to false)
- INFLUXDB_V2_PROFILERS - set the list of enabled Flux profilers

```
self.client = InfluxDBClient.from_env_properties()
```

8.3 Profile query

The Flux Profiler package provides performance profiling tools for Flux queries and operations.

You can enable printing profiler information of the Flux query in client library by:

- set QueryOptions.profilers in QueryApi,
- set INFLUXDB_V2_PROFILERS environment variable,
- set profilers option in configuration file.

When the profiler is enabled, the result of flux query contains additional tables "profiler/*". In order to have consistent behaviour with enabled/disabled profiler, FluxCSVParser excludes "profiler/*" measurements from result.

Example how to enable profilers using API:

```
q = '''
    from(bucket: stringParam)
      |> range(start: -5m, stop: now())
      |> filter(fn: (r) => r._measurement == "mem")
      |> filter(fn: (r) => r._field == "available" or r._field == "free" or r._field
↪== "used")
      |> aggregateWindow(every: 1m, fn: mean)
      |> pivot(rowKey:["_time"], columnKey: ["_field"], valueColumn: "_value")
'''
p = {
    "stringParam": "my-bucket",
}

query_api = client.query_api(query_options=QueryOptions(profilers=["query", "operator
↪"]))
csv_result = query_api.query(query=q, params=p)
```

Example of a profiler output:

You can also use callback function to get profilers output. Return value of this callback is type of FluxRecord.

Example how to use profilers with callback:

```
 class ProfilersCallback(object):
    def __init__(self):
        self.records = []

    def __call__(self, flux_record):
        self.records.append(flux_record.values)

callback = ProfilersCallback()

query_api = client.query_api(query_options=QueryOptions(profilers=["query", "operator
↪"], profiler_callback=callback))
tables = query_api.query('from(bucket:"my-bucket") |> range(start: -10m)')

for profiler in callback.records:
    print(f'Custom processing of profiler result: {profiler}')
```

Example output of this callback:

CHAPTER 9

Indices and tables

- genindex
- modindex
- search

Index

E

F

G

H

I

L

M

N

O

V

W