

---

# **influxdb\_client**

***Release 1.39.0***

**Robert Hajek, Jakub Bednar**

**Dec 05, 2023**



## CONTENTS:

<b>1</b>	<b>User Guide</b>	<b>1</b>
1.1	Query	2
1.2	Write	3
1.2.1	The data could be written as	3
1.2.2	Batching	3
1.2.3	Default Tags	6
1.2.4	Synchronous client	7
1.3	Delete data	7
1.4	Pandas DataFrame	8
1.5	How to use Asyncio	9
1.5.1	Async APIs	9
1.5.2	Async Write API	10
1.5.3	Async Query API	10
1.5.4	Async Delete API	11
1.5.5	Management API	12
1.5.6	Proxy and redirects	12
1.6	Gzip support	13
1.7	Proxy configuration	13
1.8	Authentication	13
1.8.1	Token	14
1.8.2	Username & Password	14
1.8.3	HTTP Basic	14
1.9	Nanosecond precision	14
1.10	Handling Errors	16
1.10.1	HTTP Retry Strategy	17
1.11	Logging	17
1.11.1	Debugging	18
1.12	Examples	18
1.12.1	How to efficiently import large dataset	18
1.12.2	Efficiency write data from IOT sensor	20
1.12.3	Connect to InfluxDB Cloud	22
1.12.4	How to use Jupyter + Pandas + InfluxDB 2	23
1.12.5	Other examples	24
<b>2</b>	<b>API Reference</b>	<b>25</b>
2.1	InfluxDBClient	25
2.2	QueryApi	32
2.3	WriteApi	40
2.4	BucketsApi	46
2.5	LabelsApi	49

2.6	OrganizationsApi . . . . .	50
2.7	UsersApi . . . . .	53
2.8	TasksApi . . . . .	55
2.9	InvokableScriptsApi . . . . .	61
2.10	DeleteApi . . . . .	69
2.11	Helpers . . . . .	70
<b>3</b>	<b>Async API Reference</b>	<b>75</b>
3.1	InfluxDBClientAsync . . . . .	75
3.2	QueryApiAsync . . . . .	80
3.3	WriteApiAsync . . . . .	84
3.4	DeleteApiAsync . . . . .	86
<b>4</b>	<b>Migration Guide</b>	<b>87</b>
4.1	Before You Start . . . . .	87
4.2	Content . . . . .	87
4.3	Initializing Client . . . . .	88
4.4	Creating Database/Bucket . . . . .	88
4.5	Dropping Database/Bucket . . . . .	89
4.6	Writing LineProtocol . . . . .	89
4.7	Writing Dictionary-style object . . . . .	90
4.8	Writing Structured Data . . . . .	91
4.9	Writing Pandas DataFrame . . . . .	92
4.10	Querying . . . . .	93
<b>5</b>	<b>Development</b>	<b>95</b>
5.1	tl;dr . . . . .	95
5.2	Getting Started . . . . .	95
5.3	Linting . . . . .	96
5.4	Testing . . . . .	96
5.4.1	Code Coverage . . . . .	97
5.5	Documentation . . . . .	97
<b>6</b>	<b>Documentation</b>	<b>99</b>
<b>7</b>	<b>InfluxDB 2.0 client features</b>	<b>101</b>
<b>8</b>	<b>Installation</b>	<b>103</b>
8.1	pip install . . . . .	103
8.2	Setuptools . . . . .	104
<b>9</b>	<b>Getting Started</b>	<b>105</b>
<b>10</b>	<b>Client configuration</b>	<b>107</b>
10.1	Via File . . . . .	107
10.2	Via Environment Properties . . . . .	108
10.3	Profile query . . . . .	108
<b>11</b>	<b>Indices and tables</b>	<b>113</b>
	<b>Index</b>	<b>115</b>

**USER GUIDE**

- *Query*
- *Write*
  - *The data could be written as*
  - *Batching*
  - *Default Tags*
    - \* *Via API*
    - \* *Via Configuration file*
    - \* *Via Environment Properties*
  - *Synchronous client*
- *Delete data*
- *Pandas DataFrame*
- *How to use Asyncio*
  - *Async APIs*
  - *Async Write API*
  - *Async Query API*
  - *Async Delete API*
  - *Management API*
  - *Proxy and redirects*
- *Gzip support*
- *Proxy configuration*
- *Authentication*
  - *Token*
  - *Username & Password*
  - *HTTP Basic*
- *Nanosecond precision*
- *Handling Errors*

- *HTTP Retry Strategy*
- *Logging*
  - *Debugging*
- *Examples*
  - *How to efficiently import large dataset*
  - *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](https://github.com/influxdb/influxdb-client-python/tree/master/examples).

### 1.2.2 Batching

The batching is configurable by `write_options`:

Property	Description	Default Value
<b>batch_size</b>	number of data point to collect in a batch	1000
<b>flush_interval</b>	number of milliseconds before the batch is written	1000
<b>jitter_interval</b>	the number of milliseconds to increase the batch flush interval by a random amount	0
<b>retry_interval</b>	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
<b>max_retry_time</b>	total retry timeout in milliseconds.	180_000
<b>max_retries</b>	number of max retries when write fails	5
<b>max_retry_delay</b>	maximum delay between each retry attempt in milliseconds	125_000
<b>max_close_wait</b>	maximum amount of time to wait for batches to flush when <code>.close()</code> is called	300_000
<b>exponential_base</b>	the base for the exponential retry delay, the next delay is computed using random exponential backoff as a random value within the interval <code>retry_interval * exponential_base^(attempts-1)</code> and <code>retry_interval * exponential_base^(attempts)</code> . Example for <code>retry_interval=5_000</code> , <code>exponential_base=2</code> , <code>max_retry_delay=125_000</code> , <code>total=5</code> Retry delays are random distributed values within the ranges of <code>[5_000-10_000, 10_000-20_000, 20_000-40_000, 40_000-80_000, 80_000-125_000]</code>	2

```

from datetime import datetime, timedelta

import pandas as pd
import reactivex as rx
from reactivex 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:

        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,
                                                            max_close_wait=300_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"])

```

(continues on next page)



(continued from previous page)

```

"""
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)])

"""
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.utcnow()

```

(continues on next page)

(continued from previous page)

```

_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")

```

```

[influx2]
url=http://localhost:8086
org=my-org
token=my-token
timeout=6000

[tags]
id = 132-987-655

```

(continues on next page)

(continued from previous page)

```
customer = California Miner
data_center = ${env.data_center}
```

You can also use a [TOML](#) or a [JSON](#) 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 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 Delete data

The `delete_api.py` supports deletes [points](#) from an InfluxDB bucket.

```
from influxdb_client import InfluxDBClient

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=
```

(continues on next page)

(continued from previous page)

```
↪ 'my-org')

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

## 1.4 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 than one table then the client generates a DataFrame for each of them.

---

The client is able to retrieve data in [Pandas DataFrame](#) format through `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:

	result	table	location	temperature
0	_result	0	New York	24.3
1	_result	1	Prague	25.3

## 1.5 How to use Asyncio

Starting from version 1.27.0 for Python 3.7+ the influxdb-client package supports `async/await` based on [asyncio](#), [aiohttp](#) and [aiocsv](#). You can install `aiohttp` and `aiocsv` directly:

```
$ python -m pip install influxdb-client aiohttp aiocsv
```

or use the `[async]` extra:

```
$ python -m pip install influxdb-client[async]
```

**Warning:** The `InfluxDBClientAsync` should be initialised inside `async` coroutine otherwise there can be unexpected behaviour. For more info see: [Why is creating a ClientSession outside of an event loop dangerous?](#).

### 1.5.1 Async APIs

All async APIs are available via `InfluxDBClientAsync`. The async version of the client supports following asynchronous APIs:

- `WriteApiAsync`
- `QueryApiAsync`
- `DeleteApiAsync`
- Management services into `influxdb_client.service` supports async operation

and also check to readiness of the InfluxDB via `/ping` endpoint:

```
import asyncio

from influxdb_client.client.influxdb_client_async import InfluxDBClientAsync

async def main():
    async with InfluxDBClientAsync(url="http://localhost:8086", token="my-token",
    ↪ org="my-org") as client:
        ready = await client.ping()
        print(f"InfluxDB: {ready}")

if __name__ == "__main__":
    asyncio.run(main())
```

## 1.5.2 Async Write API

The *WriteApiAsync* supports ingesting data as:

- string or bytes that is formatted as a InfluxDB's line protocol
- Data Point structure
- Dictionary style mapping with keys: measurement, tags, fields and time or custom structure
- NamedTuple
- Data Classes
- Pandas DataFrame
- List of above items

```
import asyncio

from influxdb_client import Point
from influxdb_client.client.influxdb_client_async import InfluxDBClientAsync

async def main():
    async with InfluxDBClientAsync(url="http://localhost:8086", token="my-token", org="my-org") as client:

        write_api = client.write_api()

        _point1 = Point("async_m").tag("location", "Prague").field("temperature", 25.3)
        _point2 = Point("async_m").tag("location", "New York").field("temperature", 24.3)

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

        print(f" > successfully: {successfully}")

if __name__ == "__main__":
    asyncio.run(main())
```

## 1.5.3 Async Query API

The *QueryApiAsync* supports retrieve data as:

- List of *FluxTable*
- Stream of *FluxRecord* via *AsyncGenerator*
- Pandas DataFrame
- Stream of Pandas DataFrame via *AsyncGenerator*
- Raw str output

```

import asyncio

from influxdb_client.client.influxdb_client_async import InfluxDBClientAsync

async def main():
    async with InfluxDBClientAsync(url="http://localhost:8086", token="my-token",
    ↪, org="my-org") as client:
        # Stream of FluxRecords
        query_api = client.query_api()
        records = await query_api.query_stream('from(bucket:"my-bucket") '
        '|> range(start: -10m) '
        '|> filter(fn: (r) => r["_
    ↪measurement"] == "async_m")')
        async for record in records:
            print(record)

if __name__ == "__main__":
    asyncio.run(main())

```

### 1.5.4 Async Delete API

```

import asyncio
from datetime import datetime

from influxdb_client.client.influxdb_client_async import InfluxDBClientAsync

async def main():
    async with InfluxDBClientAsync(url="http://localhost:8086", token="my-token",
    ↪, org="my-org") as client:
        start = datetime.utcnow()
        stop = datetime.now()
        # Delete data with location = 'Prague'
        successfully = await client.delete_api().delete(start=start, stop=stop,
    ↪ bucket="my-bucket",
        predicate="location = \
    ↪\"Prague\"")
        print(f" > successfully: {successfully}")

if __name__ == "__main__":
    asyncio.run(main())

```

### 1.5.5 Management API

```
import asyncio

from influxdb_client import OrganizationsService
from influxdb_client.client.influxdb_client_async import InfluxDBClientAsync

async def main():
    async with InfluxDBClientAsync(url='http://localhost:8086', token='my-token',
    ↪, org='my-org') as client:
        # Initialize async OrganizationsService
        organizations_service = OrganizationsService(api_client=client.api_
    ↪client)

        # Find organization with name 'my-org'
        organizations = await organizations_service.get_orgs(org='my-org')
        for organization in organizations.orgs:
            print(f'name: {organization.name}, id: {organization.id}')

if __name__ == "__main__":
    asyncio.run(main())
```

### 1.5.6 Proxy and redirects

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.client.influxdb_client_async import InfluxDBClientAsync

async with InfluxDBClientAsync(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.

---

Client automatically follows HTTP redirects. The default redirect policy is to follow up to 10 consecutive requests. The redirects can be configured via:

- `allow_redirects` - If set to `False`, do not follow HTTP redirects. `True` by default.
- `max_redirects` - Maximum number of HTTP redirects to follow. 10 by default.



## 1.6 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)
```

## 1.7 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
```

## 1.8 Authentication

InfluxDBClient supports three options how to authorize a connection:

- *Token*
- *Username & Password*
- *HTTP Basic*

### 1.8.1 Token

Use the `token` to authenticate to the InfluxDB API. In your API requests, an *Authorization* header will be send. The header value, provide the word *Token* followed by a space and an InfluxDB API token. The word *token* is case-sensitive.

```
from influxdb_client import InfluxDBClient

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

---

**Note:** Note that this is a preferred way how to authenticate to InfluxDB API.

---

### 1.8.2 Username & Password

Authenticates via username and password credentials. If successful, creates a new session for the user.

```
from influxdb_client import InfluxDBClient

with InfluxDBClient(url="http://localhost:8086", username="my-user", password="my-
↳password") as client
```

**Warning:** The username/password auth is based on the HTTP “Basic” authentication. The authorization expires when the *time-to-live (TTL)* (default 60 minutes) is reached and client produces *unauthorized* exception.

### 1.8.3 HTTP Basic

Use this 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.

```
from influxdb_client import InfluxDBClient

with InfluxDBClient(url="http://localhost:8086", auth_basic=True, token="my-proxy-secret
↳") as client
```

**Warning:** Don't use this when directly talking to InfluxDB 2.

## 1.9 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:
    print(f'Temperature in {record["location"]} is {record["_value"]} at time: {record["_time"]}')

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

```

## 1.10 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()
    with client.write_api(success_callback=callback.success,
                          error_callback=callback.error,
                          retry_callback=callback.retry) as write_api:
        pass
```

### 1.10.1 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)
```

## 1.11 Logging

The client uses Python's [logging](#) facility for logging the library activity. The following logger categories are exposed:

- `influxdb_client.client.influxdb_client`
- `influxdb_client.client.influxdb_client_async`
- `influxdb_client.client.write_api`
- `influxdb_client.client.write_api_async`
- `influxdb_client.client.write.retry`
- `influxdb_client.client.write.dataframe_serializer`
- `influxdb_client.client.util.multiprocessing_helper`
- `influxdb_client.client.http`
- `influxdb_client.client.exceptions`

The default logging level is *warning* without configured logger output. You can use the standard logger interface to change the log level and handler:

```
import logging
import sys

from influxdb_client import InfluxDBClient

with InfluxDBClient(url="http://localhost:8086", token="my-token", org="my-org") as
↳ client:
    for _, logger in client.conf.loggers.items():
        logger.setLevel(logging.DEBUG)
        logger.addHandler(logging.StreamHandler(sys.stdout))
```

### 1.11.1 Debugging

For debug purpose you can enable verbose logging of HTTP requests and set the debug level to all client's logger categories by:

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

---

**Note:** Both HTTP request headers and body will be logged to standard output.

---

## 1.12 Examples

### 1.12.1 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 reactivex as rx
from reactivex import operators as ops

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,
↪1198195200000000000000

    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
```

(continues on next page)

(continued from previous page)

```

: return: Parsed csv row to [Point]
"""

"""
    For better performance is sometimes useful directly create a LineProtocol to avoid
↳ unnecessary escaping overhead:
    """
    # from datetime import timezone
    # import ciso8601
    # from influxdb_client.client.write.point import EPOCH
    #
    # time = (ciso8601.parse_datetime(row["Date"]).replace(tzinfo=timezone.utc) -
↳ 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)

"""
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
"""

```

(continues on next page)

(continued from previous page)

```

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()

```

## 1.12.2 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 reactivex as rx
from reactivex 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.

    :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].

```

(continues on next page)



(continued from previous page)

```

        :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)

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

```

(continues on next page)

(continued from previous page)

`input()`

### 1.12.3 Connect to InfluxDB Cloud

The following example demonstrate a simplest way how to write and query data 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')
    print()
    print()

```

(continues on next page)

(continued from previous page)

```

"""
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()

```

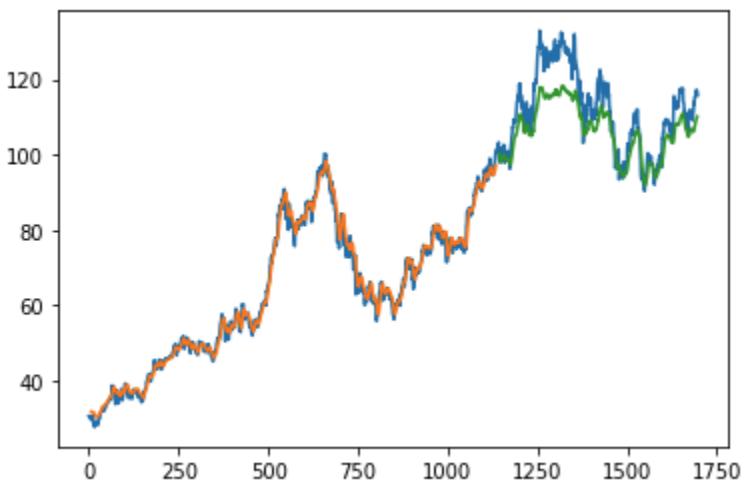
### 1.12.4 How to use Jupyter + Pandas + InfluxDB 2

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](#)

### 1.12.5 Other examples

You can find all examples at GitHub: [influxdb-client-python/examples](#).

## API REFERENCE

- *InfluxDBClient*
- *QueryApi*
- *WriteApi*
- *BucketsApi*
- *LabelsApi*
- *OrganizationsApi*
- *UsersApi*
- *TasksApi*
- *InvokableScriptsApi*
- *DeleteApi*
- *Helpers*

### 2.1 InfluxDBClient

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

InfluxDBClient is client for InfluxDB v2.

Initialize defaults.

#### Parameters

- **url** – InfluxDB server API url (ex. <http://localhost:8086>).
- **token** – token to authenticate to the InfluxDB API
- **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, Write and Delete 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 cert\_file**

Path to the certificate that will be used for mTLS authentication.

**Key str cert\_key\_file**

Path to the file contains private key for mTLS certificate.

**Key str cert\_key\_password**

String or function which returns password for decrypting the mTLS private key.

**Key ssl.SSLContext ssl\_context**

Specify a custom Python SSL Context for the TLS/ mTLS handshake. Be aware that only delivered certificate/ key files or an SSL Context are possible.

**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 str username**

username to authenticate via username and password credentials to the InfluxDB 2.x

**Key str password**

password to authenticate via username and password credentials to the InfluxDB 2.x

**Key list[str] profilers**

list of enabled Flux profilers

**authorizations\_api()** → AuthorizationsApi

Create the Authorizations API instance.

**Returns**

authorizations api

**buckets\_api()** → *BucketsApi*

Create the Bucket API instance.

**Returns**

buckets api

**build()** → *str*

Return the build type of the connected InfluxDB Server.

**Returns**

The type of InfluxDB build.

**close()**

Shutdown the 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, \*\*kwargs*)

Configure client via configuration file. The configuration has to be under 'influx' section.

**Parameters**

- **config\_file** – Path to configuration file
- **debug** – Enable verbose logging of http requests
- **enable\_gzip** – Enable Gzip compression for http requests. Currently, only the “Write” and “Query” endpoints supports the Gzip compression.

**Key config\_name**

Name of the configuration section of the configuration file

**Key str proxy\_headers**

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

**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 ssl.SSLContext ssl\_context**

Specify a custom Python SSL Context for the TLS/ mTLS handshake. Be aware that only delivered certificate/ key files or an SSL Context are possible.

**The supported formats:**

- <https://docs.python.org/3/library/configparser.html>
- <https://toml.io/en/>
- <https://www.json.org/json-en.html>

**Configuration options:**

- url
- org
- token
- timeout,
- verify\_ssl
- ssl\_ca\_cert

- cert\_file
- cert\_key\_file
- cert\_key\_password
- 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}"
```

config.json example:

```
{
  "url": "http://localhost:8086",
  "token": "my-token",
  "org": "my-org",
  "active": true,
  "timeout": 6000,
  "connection_pool_maxsize": 55,
```

(continues on next page)



(continued from previous page)

```

    "auth_basic": false,
    "profilers": "query, operator",
    "tags": {
        "id": "132-987-655",
        "customer": "California Miner",
        "data_center": "${env.data_center}"
    }
}

```

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

Configure client via environment properties.

#### Parameters

- **debug** – Enable verbose logging of http requests
- **enable\_gzip** – Enable Gzip compression for http requests. Currently, only the “Write” and “Query” endpoints supports the Gzip compression.

#### 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 `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 `ssl.SSLContext` **ssl\_context**

Specify a custom Python SSL Context for the TLS/ mTLS handshake. Be aware that only delivered certificate/ key files or an SSL Context are possible.

#### 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\_CERT\_FILE
- INFLUXDB\_V2\_CERT\_KEY\_FILE
- INFLUXDB\_V2\_CERT\_KEY\_PASSWORD
- INFLUXDB\_V2\_CONNECTION\_POOL\_MAXSIZE
- INFLUXDB\_V2\_AUTH\_BASIC
- INFLUXDB\_V2\_PROFILERS
- INFLUXDB\_V2\_TAG

**health()** → *HealthCheck*

Get the health of an instance.

**Returns**

*HealthCheck*

**invokable\_scripts\_api()** → *InvokableScriptsApi*

Create an InvokableScripts API instance.

**Returns**

*InvokableScripts* API instance

**labels\_api()** → *LabelsApi*

Create the Labels API instance.

**Returns**

labels api

**organizations\_api()** → *OrganizationsApi*

Create the Organizations API instance.

**Returns**

organizations api

**ping()** → *bool*

Return 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>) → *QueryApi*

Create an Query API instance.

**Parameters**

**query\_options** – optional query api configuration

**Returns**

Query api instance

**ready()** → *Ready*

Get The readiness of the InfluxDB 2.0.

**Returns**

*Ready*

**tasks\_api()** → *TasksApi*

Create the Tasks API instance.

**Returns**

tasks api

**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*) → *WriteApi*

Create Write API instance.

**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)
```

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 written 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 written 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: *Optional[dict]* = None) → *TableList*

Execute synchronous Flux query and return result as a *FluxTable* list.

**Parameters**

- **query** – the Flux query

- **org** (*str*, *Organization*) – specifies the organization for executing the query; Take the ID, Name or Organization. If not specified the default value from `InfluxDBClient.org` is used.
- **params** – bind parameters

**Returns**

*FluxTable* list wrapped into *TableList*

**Return type**

*TableList*

Serialization the query results to flattened list of values via `to_values()`:

```
from influxdb_client import InfluxDBClient

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

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

    # Serialize to values
    output = tables.to_values(columns=['location', '_time', '_value'])
    print(output)
```

```
[
  ['New York', datetime.datetime(2022, 6, 7, 11, 3, 22, 917593, tzinfo=tzutc()), 24.3],
  ['Prague', datetime.datetime(2022, 6, 7, 11, 3, 22, 917593, tzinfo=tzutc()), 25.3],
  ...
]
```

Serialization the query results to JSON via `to_json()`:

```
from influxdb_client import InfluxDBClient

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

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

    # Serialize to JSON
    output = tables.to_json(indent=5)
    print(output)
```

```
[
  {
    "_measurement": "mem",
    "_start": "2021-06-23T06:50:11.897825+00:00",
    "_stop": "2021-06-25T06:50:11.897825+00:00",
```

(continues on next page)

(continued from previous page)

```

        "_time": "2020-02-27T16:20:00.897825+00:00",
        "region": "north",
        "_field": "usage",
        "_value": 15
    },
    {
        "_measurement": "mem",
        "_start": "2021-06-23T06:50:11.897825+00:00",
        "_stop": "2021-06-25T06:50:11.897825+00:00",
        "_time": "2020-02-27T16:20:01.897825+00:00",
        "region": "west",
        "_field": "usage",
        "_value": 10
    },
    ...
]

```

**query\_csv**(query: *str*, org=None, dialect: *Dialect* = {'annotations': ['datatype', 'group', 'default'], 'comment\_prefix': '#', 'date\_time\_format': 'RFC3339', 'delimiter': ',', 'header': True}, params: *Optional[dict]* = None) → *CSVIterator*

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
- **org** (*str*, *Organization*) – specifies the organization for executing the query; Take the ID, Name or Organization. If not specified the default value from `InfluxDBClient.org` is used.
- **dialect** – csv dialect format
- **params** – bind parameters

#### Returns

`Iterator[List[str]]` wrapped into *CSVIterator*

#### Return type

*CSVIterator*

Serialization the query results to flattened list of values via `to_values()`:

```

from influxdb_client import InfluxDBClient

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

    # Query: using CSV iterator
    csv_iterator = client.query_api().query_csv('from(bucket:"my-bucket") |>
    range(start: -10m)')

    # Serialize to values
    output = csv_iterator.to_values()
    print(output)

```

```
[
    ['#datatype', 'string', 'long', 'dateTime:RFC3339', 'dateTime:RFC3339',
     ↪ 'dateTime:RFC3339', 'double', 'string', 'string', 'string']
    ['#group', 'false', 'false', 'true', 'true', 'false', 'false', 'true', 'true'
     ↪ ', 'true']
    ['#default', '_result', '', '', '', '', '', '', '']
    ['', 'result', 'table', '_start', '_stop', '_time', '_value', '_field', '_'
     ↪ 'measurement', 'location']
    ['', '', '0', '2022-06-16', '2022-06-16', '2022-06-16', '24.3', 'temperature'
     ↪ ', 'my_measurement', 'New York']
    ['', '', '1', '2022-06-16', '2022-06-16', '2022-06-16', '25.3', 'temperature'
     ↪ ', 'my_measurement', 'Prague']
    ...
]
```

If you would like to turn off Annotated CSV header's you can use following code:

```
from influxdb_client import InfluxDBClient, Dialect

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

    # Query: using CSV iterator
    csv_iterator = client.query_api().query_csv('from(bucket:"my-bucket") |>_
↪range(start: -10m)',
                                                dialect=Dialect(header=False,
↪
↪annotations=[]))

    for csv_line in csv_iterator:
        print(csv_line)
```

```
[
    ['', '_result', '0', '2022-06-16', '2022-06-16', '2022-06-16', '24.3',
     ↪ 'temperature', 'my_measurement', 'New York']
    ['', '_result', '1', '2022-06-16', '2022-06-16', '2022-06-16', '25.3',
     ↪ 'temperature', 'my_measurement', 'Prague']
    ...
]
```

**query\_data\_frame**(query: str, org=None, data\_frame\_index: Optional[List[str]] = None, params: Optional[dict] = None)

Execute synchronous Flux query and return Pandas DataFrame.

**Note:** If the query returns tables with differing schemas than the client generates a DataFrame for each of them.

#### Parameters

- **query** – the Flux query
- **org** (str, Organization) – specifies the organization for executing the query; Take the ID, Name or Organization. If not specified the default value from InfluxDBClient.org is used.

- **data\_frame\_index** – the list of columns that are used as DataFrame index
- **params** – bind parameters

**Returns**

DataFrame or List[DataFrame]

**Warning:** For the optimal processing of the query results use the `pivot()` function which align results as a table.

```
from(bucket:"my-bucket")
  |> range(start: -5m, stop: now())
  |> filter(fn: (r) => r._measurement == "mem")
  |> pivot(rowKey:["_time"], columnKey: ["_field"], valueColumn: "_value")
```

**For more info see:**

- <https://docs.influxdata.com/resources/videos/pivots-in-flux/>
- <https://docs.influxdata.com/flux/latest/stdlib/universe/pivot/>
- <https://docs.influxdata.com/flux/latest/stdlib/influxdata/influxdb/schema/fieldsascols/>

**query\_data\_frame\_stream**(*query: str, org=None, data\_frame\_index: Optional[List[str]] = None, params: Optional[dict] = None*)

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

---

**Note:** If the query returns tables with differing schemas than the client generates a DataFrame for each of them.

---

**Parameters**

- **query** – the Flux query
- **org** (*str*, [Organization](#)) – specifies the organization for executing the query; Take the ID, Name or Organization. If not specified the default value from `InfluxDBClient.org` is used.
- **data\_frame\_index** – the list of columns that are used as DataFrame index
- **params** – bind parameters

**Returns**

Generator[DataFrame]

**Warning:** For the optimal processing of the query results use the `pivot()` function which align results as a table.

```
from(bucket:"my-bucket")
  |> range(start: -5m, stop: now())
  |> filter(fn: (r) => r._measurement == "mem")
  |> pivot(rowKey:["_time"], columnKey: ["_field"], valueColumn: "_value")
```

**For more info see:**



- <https://docs.influxdata.com/resources/videos/pivots-in-flux/>
- <https://docs.influxdata.com/flux/latest/stdlib/universe/pivot/>
- <https://docs.influxdata.com/flux/latest/stdlib/influxdata/influxdb/schema/fieldsascols/>

**query\_raw**(query: *str*, org=None, dialect={'annotations': ['datatype', 'group', 'default'], 'comment\_prefix': '#', 'date\_time\_format': 'RFC3339', 'delimiter': ',', 'header': True}, params: *Optional[dict]* = None)

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

#### Parameters

- **query** – a Flux query
- **org** (*str*, *Organization*) – specifies the organization for executing the query; Take the ID, Name or Organization. If not specified the default value from `InfluxDBClient.org` is used.
- **dialect** – csv dialect format
- **params** – bind parameters

#### Returns

*str*

**query\_stream**(query: *str*, org=None, params: *Optional[dict]* = None) → *Generator[FluxRecord, Any, None]*

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

#### Parameters

- **query** – the Flux query
- **org** (*str*, *Organization*) – specifies the organization for executing the query; Take the ID, Name or Organization. If not specified the default value from `InfluxDBClient.org` is used.
- **params** – bind parameters

#### Returns

*Generator['FluxRecord']*

**class** influxdb\_client.client.flux\_table.FluxTable

A table is set of records with a common set of columns and a group key.

The table can be serialized into JSON by:

```
import json
from influxdb_client.client.flux_table import FluxStructureEncoder

output = json.dumps(tables, cls=FluxStructureEncoder, indent=2)
print(output)
```

Initialize defaults.

**get\_group\_key()**

Group key is a list of columns.

A table's group key denotes which subset of the entire dataset is assigned to the table.

```
class influxdb_client.client.flux_table.FluxRecord(table, values=None)
```

A record is a tuple of named values and is represented using an object type.

Initialize defaults.

```
get_field()
```

Get field name.

```
get_measurement()
```

Get measurement name.

```
get_start()
```

Get ‘\_start’ value.

```
get_stop()
```

Get ‘\_stop’ value.

```
get_time()
```

Get timestamp.

```
get_value()
```

Get field value.

```
class influxdb_client.client.flux_table.TableList(iterable=(),/)
```

*FluxTable* list with additionally functional to better handle of query result.

```
to_json(columns: Optional[List[str]] = None, **kwargs) → str
```

Serialize query results to a JSON formatted `str`.

#### Parameters

**columns** – if not None then only specified columns are presented in results

#### Returns

`str`

The query results is flattened to array:

```
[
  {
    "_measurement": "mem",
    "_start": "2021-06-23T06:50:11.897825+00:00",
    "_stop": "2021-06-25T06:50:11.897825+00:00",
    "_time": "2020-02-27T16:20:00.897825+00:00",
    "region": "north",
    "_field": "usage",
    "_value": 15
  },
  {
    "_measurement": "mem",
    "_start": "2021-06-23T06:50:11.897825+00:00",
    "_stop": "2021-06-25T06:50:11.897825+00:00",
    "_time": "2020-02-27T16:20:01.897825+00:00",
    "region": "west",
    "_field": "usage",
    "_value": 10
  },
  ...
]
```

The JSON format could be configured via **\*\*kwargs** arguments:

```
from influxdb_client import InfluxDBClient

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

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

    # Serialize to JSON
    output = tables.to_json(indent=5)
    print(output)
```

For all available options see - `json.dump`.

**to\_values**(columns: *Optional[List[str]]* = None) → List[List[object]]

Serialize query results to a flattened list of values.

#### Parameters

**columns** – if not None then only specified columns are presented in results

#### Returns

list of values

Output example:

```
[
    ['New York', datetime.datetime(2022, 6, 7, 11, 3, 22, 917593, tzinfo=tzutc()), 24.3],
    ['Prague', datetime.datetime(2022, 6, 7, 11, 3, 22, 917593, tzinfo=tzutc()), 25.3],
    ...
]
```

Configure required columns:

```
from influxdb_client import InfluxDBClient

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

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

    # Serialize to values
    output = tables.to_values(columns=['location', '_time', '_value'])
    print(output)
```

**class** influxdb\_client.client.flux\_table.CSVIterator(response: *HTTPResponse*)

Iterator[List[str]] with additionally functional to better handle of query result.

Initialize csv.reader.

**to\_values()** → List[List[str]]

Serialize query results to a flattened list of values.

**Returns**

list of values

Output example:

```
[
  ['New York', '2022-06-14T08:00:51.749072045Z', '24.3'],
  ['Prague', '2022-06-14T08:00:51.749072045Z', '25.3'],
  ...
]
```

## 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 written 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 written 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: Optional[str] = None, record: Optional[Union[str, Iterable[str], Point, Iterable[Point], dict, Iterable[dict], bytes, Iterable[bytes], Observable, NamedTuple, Iterable[NamedTuple], dataclass, Iterable[dataclass]]] = None, write\_precision: WritePrecision = 'ns', \*\*kwargs*) → Any

Write time-series data into InfluxDB.

#### Parameters

- **bucket** (*str*) – specifies the destination bucket for writes (required)
- **org** (*str*, *Organization*) – specifies the destination organization for writes; take the ID, Name or Organization. If not specified the default value from `InfluxDBClient.org` is used.
- **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 `data_frame_timestamp_column`

name of `DataFrame` column which contains a timestamp. The column can be defined as a *str* value formatted as `2018-10-26`, `2018-10-26 12:00`, `2018-10-26 12:00:00-05:00` or other formats and types supported by `pandas.to_datetime` - `DataFrame`

#### Key `data_frame_timestamp_timezone`

name of the timezone which is used for timestamp column - `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")

# 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:**

If the `data_frame_timestamp_column` is not specified the index of `Pandas DataFrame` is used as a timestamp for written data. The index can 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: <https://docs.influxdata.com/influxdb/latest/reference/key-concepts/data-elements/#point>

Initialize defaults.

**field**(*field*, *value*)

Add field with key and value.

**static from\_dict**(*dictionary*: *dict*, *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"},
    "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"])
```

**Int Types:**

The following example shows how to configure the types of integers fields. It is useful when you want to serialize integers always as float to avoid field type conflict or use unsigned 64-bit integer as the type for serialization.

```
# Use custom dictionary structure
dict_structure = {
    "measurement": "h2o_feet",
    "tags": {"location": "coyote_creek"},
    "fields": {
        "water_level": 1.0,
        "some_counter": 108913123234
    }
}
```

(continues on next page)

(continued from previous page)

```

    },
    "time": 1
}

point = Point.from_dict(dict_structure, field_types={"some_counter": "uint"}
↪)

```

**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

**Key field\_types**

optional dictionary to specify types of serialized fields. Currently, is supported customization for integer types. Possible integers types:

- **int** - serialize integers as “**Signed 64-bit integers**” - 9223372036854775807i (default behaviour)
- **uint** - serialize integers as “**Unsigned 64-bit integers**” - 9223372036854775807u
- **float** - serialize integers as “**IEEE-754 64-bit floating-point numbers**”. Useful for unify number types in your pipeline to avoid field type conflict - 9223372036854775807

The **field\_types** can be also specified as part of incoming dictionary. For more info see an example above.

**Returns**

new data point

**static measurement**(*measurement*)

Create a new Point with specified measurement name.

**classmethod set\_str\_rep**(*rep\_function*)

Set the string representation for all Points.

**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**(*precision=None*)

Create LineProtocol.

**param precision**

required precision of LineProtocol. If it's not set then use the precision from Point.

**property 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*) → *Bucket*

Create a bucket.

### Parameters

- **bucket** (*Bucket* / *PostBucketRequest*) – 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*
- **org** (*str*, *Organization*) – specifies the organization for create the bucket; Take the ID, Name or Organization. If not specified the default value from *InfluxDBClient.org* is used.

### 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*: [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.

**property created\_at**

Get the created\_at of this Bucket.

**Returns**

The created\_at of this Bucket.

**Return type**

datetime

**property description**

Get the description of this Bucket.

**Returns**

The description of this Bucket.

**Return type**

str

**property id**

Get the id of this Bucket.

**Returns**

The id of this Bucket.

**Return type**`str`**property labels**

Get the labels of this Bucket.

**Returns**

The labels of this Bucket.

**Return type**`list[Label]`**property links**

Get the links of this Bucket.

**Returns**

The links of this Bucket.

**Return type**`BucketLinks`**property name**

Get the name of this Bucket.

**Returns**

The name of this Bucket.

**Return type**`str`**property org\_id**

Get the org\_id of this Bucket.

**Returns**

The org\_id of this Bucket.

**Return type**`str`**property retention\_rules**

Get the retention\_rules of this Bucket.

Retention rules to expire or retain data. The InfluxDB */api/v2* API uses *RetentionRules* to configure the [retention period](<https://docs.influxdata.com/influxdb/latest/reference/glossary/#retention-period>). ##### InfluxDB Cloud - *retentionRules* is required. ##### InfluxDB OSS - *retentionRules* isn't required.

**Returns**

The retention\_rules of this Bucket.

**Return type**`list[BucketRetentionRules]`**property rp**

Get the rp of this Bucket.

**Returns**

The rp of this Bucket.

**Return type**`str`

**property 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.

**property type**

Get the type of this Bucket.

**Returns**

The type of this Bucket.

**Return type**

str

**property 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: 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: Optional[Dict[str, str]] = None*) → 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, 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[Label]*

Get the list of all labels for given organization.

**Parameters**

**org\_id** – organization id

**Returns**

list of labels

**find\_labels**(*\*\*kwargs*) → *List[Label]*

Get all available labels.

**Key str org\_id**

The organization ID.

**Returns**

labels

**update\_label**(*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*: *Optional[str] = None*, *organization*: *Optional[Organization] = None*) → *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: 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,
                                          default_storage_type=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.

**property created\_at**

Get the created\_at of this Organization.

**Returns**

The created\_at of this Organization.

**Return type**

datetime

**property default\_storage\_type**

Get the default\_storage\_type of this Organization.

Discloses whether the organization uses TSM or IOx.

**Returns**

The default\_storage\_type of this Organization.

**Return type**

str

**property description**

Get the description of this Organization.

**Returns**

The description of this Organization.

**Return type**

str

**property id**

Get the id of this Organization.

**Returns**

The id of this Organization.

**Return type**

str

**property links**

Get the links of this Organization.

**Returns**

The links of this Organization.

**Return type**

OrganizationLinks

**property name**

Get the name of this Organization.

**Returns**

The name of this Organization.

**Return type**

str

**property 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.

**property 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*) → *User*

Create a user.

**delete\_user**(user: *Union[str, User, UserResponse]*) → *None*

Delete a user.

**Parameters**

**user** – user id or User

**Returns**

*None*

**find\_users**(\*\*kwargs) → *Users*

List all users.

**Key int offset**

The offset for pagination. The number of records to skip.

**Key int limit**

Limits the number of records returned. Default is 20.

**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**() → *User*

Return the current authenticated user.

**update\_password**(user: *Union[str, User, UserResponse]*, password: *str*) → *None*

Update a password.

**Parameters**

- **user** – User to update password (required)
- **password** – New password (required)

**Returns**

None

**update\_user**(*user*: User) → UserResponse

Update a user.

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

User

**class** influxdb\_client.domain.User(*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.

**property id**

Get the id of this User.

The user ID.

**Returns**

The id of this User.

**Return type**

str

**property name**

Get the name of this User.

The user name.

**Returns**

The name of this User.

**Return type**

str

**property status**

Get the status of this User.

If *inactive*, the user is inactive. Default is *active*.**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*) → 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: Task) → Task

Clone a task.

**create\_task**(task: Optional[Task] = None, task\_create\_request: Optional[TaskCreateRequest] = None) → Task

Create a new task.

**create\_task\_cron**(name: *str*, flux: *str*, cron: *str*, org\_id: *str*) → Task

Create a new task with cron repetition schedule.

**create\_task\_every**(name, flux, every, organization) → 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) → Task

Retrieve a task.

**find\_tasks**(\*\*kwargs)

List all tasks up to set limit (max 500).

### 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.

**find\_tasks\_iter**(*\*\*kwargs*)

Iterate over all tasks with pagination.

**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 in one page

**Returns**

Tasks iterator

**get\_labels**(*task\_id*)

List all labels for a task.

**get\_logs**(*task\_id: str*) → [List](#)[[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*) → *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*[*LogEvent*]

Retrieve all logs for a run.

**get\_runs**(*task\_id*, *\*\*kwargs*) → *List*[*Run*]

Retrieve list of run records for a task.

**Parameters**

**task\_id** – task id

**Key str after**

returns runs after specified ID

**Key int limit**

the number of runs to return

**Key datetime after\_time**

filter runs to those scheduled after this time, RFC3339

**Key datetime before\_time**

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/asdf/installs/python/3.7.17/lib/python3.7/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*: *Task*) → *Task*

Update a task.

**update\_task\_request**(*task\_id*, *task\_update\_request*: *TaskUpdateRequest*) → *Task*

Update a task.

```
class influxdb_client.domain.Task(id=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.

**property authorization\_id**

Get the authorization\_id of this Task.

An authorization ID. Specifies the authorization used when the task communicates with the query engine. To find an authorization ID, use the [*GET /api/v2/authorizations* endpoint](#operation/GetAuthorizations) to list authorizations.

**Returns**

The authorization\_id of this Task.

**Return type**

str

**property created\_at**

Get the created\_at of this Task.

**Returns**

The created\_at of this Task.

**Return type**

datetime

**property cron**

Get the cron of this Task.

A [Cron expression](<https://en.wikipedia.org/wiki/Cron#Overview>) that defines the schedule on which the task runs. InfluxDB uses the system time when evaluating Cron expressions.

**Returns**

The cron of this Task.

**Return type**

str

**property description**

Get the description of this Task.

A description of the task.

**Returns**

The description of this Task.

**Return type**

str

**property every**

Get the every of this Task.

The interval ([duration literal](<https://docs.influxdata.com/influxdb/latest/reference/glossary/#rfc3339-timestamp>)) at which the task runs. *every* also determines when the task first runs, depending on the specified time.

**Returns**

The every of this Task.

**Return type**`str`**property flux**

Get the flux of this Task.

The Flux script that the task executes.

**Returns**

The flux of this Task.

**Return type**`str`**property id**

Get the id of this Task.

**Returns**

The id of this Task.

**Return type**`str`**property labels**

Get the labels of this Task.

**Returns**

The labels of this Task.

**Return type**`list[Label]`**property last\_run\_error**

Get the last\_run\_error of this Task.

**Returns**

The last\_run\_error of this Task.

**Return type**`str`**property last\_run\_status**

Get the last\_run\_status of this Task.

**Returns**

The last\_run\_status of this Task.

**Return type**`str`**property latest\_completed**

Get the latest\_completed of this Task.

A timestamp ([RFC3339 date/time format](<https://docs.influxdata.com/influxdb/latest/reference/glossary/#rfc3339-timestamp>)) of the latest scheduled and completed run.

**Returns**

The latest\_completed of this Task.

**Return type**`datetime`

**property links**

Get the links of this Task.

**Returns**

The links of this Task.

**Return type**

TaskLinks

**property name**

Get the name of this Task.

The name of the task.

**Returns**

The name of this Task.

**Return type**

str

**property offset**

Get the offset of this Task.

A [duration](<https://docs.influxdata.com/flux/v0.x/spec/lexical-elements/#duration-literals>) to delay execution of the task after the scheduled time has elapsed. 0 removes the offset.

**Returns**

The offset of this Task.

**Return type**

str

**property org**

Get the org of this Task.

An [organization](<https://docs.influxdata.com/influxdb/latest/reference/glossary/#organization>) name. Specifies the organization that owns the task.

**Returns**

The org of this Task.

**Return type**

str

**property org\_id**

Get the org\_id of this Task.

An [organization](<https://docs.influxdata.com/influxdb/latest/reference/glossary/#organization>) ID. Specifies the organization that owns the task.

**Returns**

The org\_id of this Task.

**Return type**

str

**property owner\_id**

Get the owner\_id of this Task.

A [user](<https://docs.influxdata.com/influxdb/latest/reference/glossary/#user>) ID. Specifies the owner of the task. To find a user ID, you can use the [*GET /api/v2/users* endpoint](#operation/GetUsers) to list users.



**Returns**

The owner\_id of this Task.

**Return type**

str

**property 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.

**property updated\_at**

Get the updated\_at of this Task.

**Returns**

The updated\_at of this Task.

**Return type**

datetime

## 2.9 InvokableScriptsApi

**class** influxdb\_client.InvokableScriptsApi(*influxdb\_client*)

Use API invokable scripts to create custom InfluxDB API endpoints that query, process, and shape data.

Initialize defaults.

**create\_script**(*create\_request*: ScriptCreateRequest) → Script

Create a script.

**Parameters**

**create\_request** (ScriptCreateRequest) – The script to create. (required)

**Returns**

The created script.

**delete\_script**(*script\_id*: str) → None

Delete a script.

**Parameters**

**script\_id** (str) – The ID of the script to delete. (required)

**Returns**

None

**find\_scripts**(\*\*kwargs)

List scripts.

**Key int limit**

The number of scripts to return.

**Key int offset**

The offset for pagination.

**Returns**

List of scripts.

**Return type**

`list[Script]`

**invoke\_script**(*script\_id*: *str*, *params*: *Optional[dict] = None*) → *TableList*

Invoke synchronously a script and return result as a *TableList*.

The bind parameters referenced in the script are substitutes with *params* key-values sent in the request body.

**Parameters**

- **script\_id** (*str*) – The ID of the script to invoke. (required)
- **params** – bind parameters

**Returns**

*FluxTable* list wrapped into *TableList*

**Return type**

*TableList*

Serialization the query results to flattened list of values via *to\_values()*:

```
from influxdb_client import InfluxDBClient

with InfluxDBClient(url="https://us-west-2-1.aws.cloud2.influxdata.com", token=
    ↪ "my-token", org="my-org") as client:

    # Query: using Table structure
    tables = client.invokable_scripts_api().invoke_script(script_id="script-id")

    # Serialize to values
    output = tables.to_values(columns=['location', '_time', '_value'])
    print(output)
```

```
[
    ['New York', datetime.datetime(2022, 6, 7, 11, 3, 22, 917593,
    ↪ tzinfo=tzutc()), 24.3],
    ['Prague', datetime.datetime(2022, 6, 7, 11, 3, 22, 917593, tzinfo=tzutc()),
    ↪ 25.3],
    ...
]
```

Serialization the query results to JSON via *to\_json()*:

```
from influxdb_client import InfluxDBClient

with InfluxDBClient(url="https://us-west-2-1.aws.cloud2.influxdata.com", token=
    ↪ "my-token", org="my-org") as client:
```

(continues on next page)

(continued from previous page)

```
# Query: using Table structure
tables = client.invokable_scripts_api().invoke_script(script_id="script-id")

# Serialize to JSON
output = tables.to_json(indent=5)
print(output)
```

```
[
  {
    "_measurement": "mem",
    "_start": "2021-06-23T06:50:11.897825+00:00",
    "_stop": "2021-06-25T06:50:11.897825+00:00",
    "_time": "2020-02-27T16:20:00.897825+00:00",
    "region": "north",
    "_field": "usage",
    "_value": 15
  },
  {
    "_measurement": "mem",
    "_start": "2021-06-23T06:50:11.897825+00:00",
    "_stop": "2021-06-25T06:50:11.897825+00:00",
    "_time": "2020-02-27T16:20:01.897825+00:00",
    "region": "west",
    "_field": "usage",
    "_value": 10
  },
  ...
]
```

**invoke\_script\_csv**(*script\_id*: *str*, *params*: *Optional[dict]* = None) → *CSVIterator*

Invoke synchronously a script and return result as a CSV iterator. Each iteration returns a row of the CSV file.

The bind parameters referenced in the script are substitutes with *params* key-values sent in the request body.

#### Parameters

- **script\_id** (*str*) – The ID of the script to invoke. (required)
- **params** – bind parameters

#### Returns

Iterator[List[str]] wrapped into *CSVIterator*

#### Return type

*CSVIterator*

Serialization the query results to flattened list of values via *to\_values()*:

```
from influxdb_client import InfluxDBClient

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

    # Query: using CSV iterator
```

(continues on next page)

(continued from previous page)

```

    csv_iterator = client.invokable_scripts_api().invoke_script_csv(script_id=
↪ "script-id")

```

```

    # Serialize to values
    output = csv_iterator.to_values()
    print(output)

```

```

[
    [' ', 'result', 'table', '_start', '_stop', '_time', '_value', '_field', '_
↪ measurement', 'location']
    [' ', ' ', '0', '2022-06-16', '2022-06-16', '2022-06-16', '24.3', 'temperature
↪ ', 'my_measurement', 'New York']
    [' ', ' ', '1', '2022-06-16', '2022-06-16', '2022-06-16', '25.3', 'temperature
↪ ', 'my_measurement', 'Prague']
    ...
]

```

**invoke\_script\_data\_frame**(*script\_id*: *str*, *params*: *Optional[dict]* = None, *data\_frame\_index*: *Optional[List[str]]* = None)

Invoke synchronously a script and return Pandas DataFrame.

The bind parameters referenced in the script are substitutes with *params* key-values sent in the request body.

**Note:** If the script returns tables with differing schemas than the client generates a DataFrame for each of them.

### Parameters

- **script\_id** (*str*) – The ID of the script to invoke. (required)
- **data\_frame\_index** (*List[str]*) – The list of columns that are used as DataFrame index.
- **params** – bind parameters

### Returns

DataFrame or List[DataFrame]

**Warning:** For the optimal processing of the query results use the `pivot()` function which align results as a table.

```

from(bucket:"my-bucket")
  |> range(start: -5m, stop: now())
  |> filter(fn: (r) => r._measurement == "mem")
  |> pivot(rowKey:["_time"], columnKey: ["_field"], valueColumn: "_value")

```

### For more info see:

- <https://docs.influxdata.com/resources/videos/pivots-in-flux/>
- <https://docs.influxdata.com/flux/latest/stdlib/universe/pivot/>
- <https://docs.influxdata.com/flux/latest/stdlib/influxdata/influxdb/schema/fieldsascols/>

**invoke\_script\_data\_frame\_stream**(*script\_id*: *str*, *params*: *Optional[dict] = None*, *data\_frame\_index*: *Optional[List[str]] = None*)

Invoke synchronously a script and return stream of Pandas DataFrame as a Generator['pd.DataFrame'].

The bind parameters referenced in the script are substitutes with *params* key-values sent in the request body.

**Note:** If the *script* returns tables with differing schemas than the client generates a DataFrame for each of them.

#### Parameters

- **script\_id** (*str*) – The ID of the script to invoke. (required)
- **data\_frame\_index** (*List[str]*) – The list of columns that are used as DataFrame index.
- **params** – bind parameters

#### Returns

Generator[DataFrame]

**Warning:** For the optimal processing of the query results use the `pivot()` function which align results as a table.

```
from(bucket:"my-bucket")
  |> range(start: -5m, stop: now())
  |> filter(fn: (r) => r._measurement == "mem")
  |> pivot(rowKey:["_time"], columnKey: ["_field"], valueColumn: "_value")
```

#### For more info see:

- <https://docs.influxdata.com/resources/videos/pivots-in-flux/>
- <https://docs.influxdata.com/flux/latest/stdlib/universe/pivot/>
- <https://docs.influxdata.com/flux/latest/stdlib/influxdata/influxdb/schema/fieldsascols/>

**invoke\_script\_raw**(*script\_id*: *str*, *params*: *Optional[dict] = None*) → *Iterator[List[str]]*

Invoke synchronously a script and return result as raw unprocessed result as a str.

The bind parameters referenced in the script are substitutes with *params* key-values sent in the request body.

#### Parameters

- **script\_id** (*str*) – The ID of the script to invoke. (required)
- **params** – bind parameters

#### Returns

Result as a str.

**invoke\_script\_stream**(*script\_id*: *str*, *params*: *Optional[dict] = None*) → *Generator[FluxRecord, Any, None]*

Invoke synchronously a script and return result as a Generator['FluxRecord'].

The bind parameters referenced in the script are substitutes with *params* key-values sent in the request body.

#### Parameters

- **script\_id** (*str*) – The ID of the script to invoke. (required)

- **params** – bind parameters

**Returns**

Stream of FluxRecord.

**Return type**

Generator['FluxRecord']

**update\_script**(*script\_id*: *str*, *update\_request*: *ScriptUpdateRequest*) → *Script*

Update a script.

**Parameters**

- **script\_id** (*str*) – The ID of the script to update. (required)
- **update\_request** (*ScriptUpdateRequest*) – Script updates to apply (required)

**Returns**

The updated.

```
class influxdb_client.domain.Script(id=None, name=None, description=None, org_id=None,
                                     script=None, language=None, url=None, created_at=None,
                                     updated_at=None)
```

NOTE: This class is auto generated by OpenAPI Generator.

Ref: <https://openapi-generator.tech>

Do not edit the class manually.

Script - a model defined in OpenAPI.

**property created\_at**

Get the created\_at of this Script.

**Returns**

The created\_at of this Script.

**Return type**

datetime

**property description**

Get the description of this Script.

**Returns**

The description of this Script.

**Return type**

*str*

**property id**

Get the id of this Script.

**Returns**

The id of this Script.

**Return type**

*str*

**property language**

Get the language of this Script.

**Returns**

The language of this Script.

**Return type**  
ScriptLanguage

**property name**

Get the name of this Script.

**Returns**  
The name of this Script.

**Return type**  
str

**property org\_id**

Get the org\_id of this Script.

**Returns**  
The org\_id of this Script.

**Return type**  
str

**property script**

Get the script of this Script.

The script to execute.

**Returns**  
The script of this Script.

**Return type**  
str

**to\_dict()**

Return the model properties as a dict.

**to\_str()**

Return the string representation of the model.

**property updated\_at**

Get the updated\_at of this Script.

**Returns**  
The updated\_at of this Script.

**Return type**  
datetime

**property url**

Get the url of this Script.

The invocation endpoint address.

**Returns**  
The url of this Script.

**Return type**  
str

**class** influxdb\_client.domain.ScriptCreateRequest(*name=None, description=None, script=None, language=None*)

NOTE: This class is auto generated by OpenAPI Generator.

Ref: <https://openapi-generator.tech>

Do not edit the class manually.

ScriptCreateRequest - a model defined in OpenAPI.

**property description**

Get the description of this ScriptCreateRequest.

Script description. A description of the script.

**Returns**

The description of this ScriptCreateRequest.

**Return type**

`str`

**property language**

Get the language of this ScriptCreateRequest.

**Returns**

The language of this ScriptCreateRequest.

**Return type**

`ScriptLanguage`

**property name**

Get the name of this ScriptCreateRequest.

Script name. The name must be unique within the organization.

**Returns**

The name of this ScriptCreateRequest.

**Return type**

`str`

**property script**

Get the script of this ScriptCreateRequest.

The script to execute.

**Returns**

The script of this ScriptCreateRequest.

**Return type**

`str`

**to\_dict()**

Return the model properties as a dict.

**to\_str()**

Return the string representation of the model.



## 2.10 DeleteApi

**class** influxdb\_client.DeleteApi(influxdb\_client)

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

Initialize defaults.

**delete**(start: Union[str, datetime], stop: Union[str, datetime], predicate: str, bucket: str, org: Optional[Union[str, Organization]] = None) → None

Delete Time series data from InfluxDB.

### Parameters

- **start** (str, datetime.datetime) – start time
- **stop** (str, datetime.datetime) – stop time
- **predicate** (str) – predicate
- **bucket** (str) – bucket id or name from which data will be deleted
- **org** (str, Organization) – specifies the organization to delete data from. Take the ID, Name or Organization. If not specified the default value from InfluxDBClient.org is used.

### 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.

### property predicate

Get the predicate of this DeletePredicateRequest.

An expression in [delete predicate syntax](<https://docs.influxdata.com/influxdb/latest/reference/syntax/delete-predicate/>).

### Returns

The predicate of this DeletePredicateRequest.

### Return type

str

### property start

Get the start of this DeletePredicateRequest.

A timestamp ([RFC3339 date/time format](<https://docs.influxdata.com/influxdb/latest/reference/glossary/#rfc3339-timestamp>)). The earliest time to delete from.

### Returns

The start of this DeletePredicateRequest.

### Return type

datetime

**property stop**

Get the stop of this DeletePredicateRequest.

A timestamp ([RFC3339 date/time format](<https://docs.influxdata.com/influxdb/latest/reference/glossary/#rfc3339-timestamp>)). The latest time to delete from.

**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.

## 2.11 Helpers

**class** influxdb\_client.client.util.date\_utils.DateHelper(*timezone: tzinfo = datetime.timezone.utc*)

DateHelper to groups different implementations of date operations.

If you would like to serialize the query results to custom timezone, you can use following code:

```
from influxdb_client.client.util import date_utils
from influxdb_client.client.util.date_utils import DateHelper
import dateutil.parser
from dateutil import tz

def parse_date(date_string: str):
    return dateutil.parser.parse(date_string).astimezone(tz.gettz('ETC/GMT+2'))

date_utils.date_helper = DateHelper()
date_utils.date_helper.parse_date = parse_date
```

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/asdf/installs/python/3.7.17/lib/python3.7/datetime.py'>*)

Convert datetime to UTC timezone.

**Parameters****value** – datetime**Returns**

datetime in UTC

```
class influxdb_client.client.util.date_utils_pandas.PandasDateTimeHelper(timezone: tzinfo =
                                                                    date-
                                                                    time.timezone.utc)
```

DateHelper that use Pandas library with nanosecond precision.

Initialize defaults.

**Parameters**

**timezone** – Default timezone used for serialization “datetime” without “tzinfo”. Default value is “UTC”.

**parse\_date**(date\_string: str)

Parse date string into class ‘pandas.\_libs.tslibs.timestamps.Timestamp’.

**to\_nanoseconds**(delta)

Get number of nanoseconds with nanos precision.

```
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__()

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", _
```

(continues on next page)

(continued from previous page)

```

↪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}")

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`



## ASYNC API REFERENCE

- *InfluxDBClientAsync*
- *QueryApiAsync*
- *WriteApiAsync*
- *DeleteApiAsync*

### 3.1 InfluxDBClientAsync

```
class influxdb_client.client.influxdb_client_async.InfluxDBClientAsync(url, token:
                                Optional[str] = None,
                                org: Optional[str] =
                                None, debug=None,
                                timeout=10000,
                                enable_gzip=False,
                                **kwargs)
```

InfluxDBClientAsync is client for InfluxDB v2.

Initialize defaults.

#### Parameters

- **url** – InfluxDB server API url (ex. <http://localhost:8086>).
- **token** – token to authenticate to the InfluxDB 2.x
- **org** – organization name (used as a default in Query, Write and Delete API)
- **debug** – enable verbose logging of http requests
- **timeout** – The maximal number of milliseconds for the whole HTTP request including connection establishment, request sending and response reading. It can also be a `ClientTimeout` which is directly pass to `aiohttp`.
- **enable\_gzip** – Enable Gzip compression for http requests. Currently, only the “Write” and “Query” endpoints supports the Gzip compression.

#### 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 cert\_file**

Path to the certificate that will be used for mTLS authentication.

**Key str cert\_key\_file**

Path to the file contains private key for mTLS certificate.

**Key str cert\_key\_password**

String or function which returns password for decrypting the mTLS private key.

**Key ssl.SSLContext ssl\_context**

Specify a custom Python SSL Context for the TLS/ mTLS handshake. Be aware that only delivered certificate/ key files or an SSL Context are possible.

**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**

The total number of simultaneous connections. Defaults to “multiprocessing.cpu\_count() \* 5”.

**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 str username**

username to authenticate via username and password credentials to the InfluxDB 2.x

**Key str password**

password to authenticate via username and password credentials to the InfluxDB 2.x

**Key bool allow\_redirects**

If set to False, do not follow HTTP redirects. True by default.

**Key int max\_redirects**

Maximum number of HTTP redirects to follow. 10 by default.

**Key dict client\_session\_kwargs**

Additional configuration arguments for ClientSession

**Key type client\_session\_type**

Type of aiohttp client to use. Useful for third party wrappers like aiohttp-retry. ClientSession by default.

**Key list[str] profilers**

list of enabled Flux profilers

**async build() → str**

Return the build type of the connected InfluxDB Server.

**Returns**

The type of InfluxDB build.

**async close()**

Shutdown the client.

**delete\_api() → DeleteApiAsync**

Get the asynchronous delete metrics API instance.



**Returns**

delete api

**classmethod from\_config\_file**(*config\_file: str = 'config.ini', debug=None, enable\_gzip=False, \*\*kwargs*)

Configure client via configuration file. The configuration has to be under 'influx' section.

**Parameters**

- **config\_file** – Path to configuration file
- **debug** – Enable verbose logging of http requests
- **enable\_gzip** – Enable Gzip compression for http requests. Currently, only the “Write” and “Query” endpoints supports the Gzip compression.

**Key config\_name**

Name of the configuration section of the configuration file

**Key str proxy\_headers**

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

**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 ssl.SSLContext ssl\_context**

Specify a custom Python SSL Context for the TLS/ mTLS handshake. Be aware that only delivered certificate/ key files or an SSL Context are possible.

**The supported formats:**

- <https://docs.python.org/3/library/configparser.html>
- <https://toml.io/en/>
- <https://www.json.org/json-en.html>

**Configuration options:**

- url
- org
- token
- timeout,
- verify\_ssl
- ssl\_ca\_cert
- cert\_file
- cert\_key\_file
- cert\_key\_password
- 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}"
```

config.json example:

```
{
  "url": "http://localhost:8086",
  "token": "my-token",
  "org": "my-org",
  "active": true,
  "timeout": 6000,
  "connection_pool_maxsize": 55,
  "auth_basic": false,
  "profilers": "query, operator",
  "tags": {
    "id": "132-987-655",
    "customer": "California Miner",
    "data_center": "${env.data_center}"
  }
}
```

**classmethod from\_env\_properties**(*debug=None, enable\_gzip=False, \*\*kwargs*)

Configure client via environment properties.

**Parameters**

- **debug** – Enable verbose logging of http requests
- **enable\_gzip** – Enable Gzip compression for http requests. Currently, only the “Write” and “Query” endpoints supports the Gzip compression.

**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 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 ssl.SSLContext ssl\_context**

Specify a custom Python SSL Context for the TLS/ mTLS handshake. Be aware that only delivered certificate/ key files or an SSL Context are possible.

**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\_CERT\_FILE
- INFLUXDB\_V2\_CERT\_KEY\_FILE
- INFLUXDB\_V2\_CERT\_KEY\_PASSWORD
- INFLUXDB\_V2\_CONNECTION\_POOL\_MAXSIZE
- INFLUXDB\_V2\_AUTH\_BASIC
- INFLUXDB\_V2\_PROFILERS
- INFLUXDB\_V2\_TAG

**async ping()** → `bool`

Return 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>`) → `QueryApiAsync`

Create an asynchronous Query API instance.

**Parameters**

**query\_options** – optional query api configuration

**Returns**

Query api instance

**async version()** → *str*

Return the version of the connected InfluxDB Server.

**Returns**

The version of InfluxDB.

**write\_api**(*point\_settings=<influxdb\_client.client.write\_api.PointSettings object>*) → *WriteApiAsync*

Create an asynchronous Write API instance.

**Example:**

```
from influxdb_client_async import InfluxDBClientAsync

# Initialize async/await instance of Write API
async with InfluxDBClientAsync(url="http://localhost:8086", token="my-token",
    ↪, org="my-org") as client:
    write_api = client.write_api()
```

**Parameters**

**point\_settings** – settings to store default tags

**Returns**

write api instance

## 3.2 QueryApiAsync

**class** `influxdb_client.client.query_api_async.QueryApiAsync`(*influxdb\_client,*  
*query\_options=<influxdb\_client.client.query\_api.QueryOptions object>*)

Asynchronous implementation for ‘/api/v2/query’ endpoint.

Initialize query client.

**Parameters**

**influxdb\_client** – influxdb client

**async query**(*query: str, org=None, params: Optional[dict] = None*) → *TableList*

Execute asynchronous Flux query and return result as a *FluxTable* list.

**Parameters**

- **query** – the Flux query
- **org** (*str, Organization*) – specifies the organization for executing the query; Take the ID, Name or Organization. If not specified the default value from `InfluxDBClientAsync.org` is used.
- **params** – bind parameters

**Returns**

*FluxTable* list wrapped into *TableList*

**Return type**

*TableList*

Serialization the query results to flattened list of values via `to_values()`:

```

from influxdb_client import InfluxDBClient

async with InfluxDBClientAsync(url="http://localhost:8086", token="my-token",
    ↪org="my-org") as client:

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

    # Serialize to values
    output = tables.to_values(columns=['location', '_time', '_value'])
    print(output)

```

```

[
    ['New York', datetime.datetime(2022, 6, 7, 11, 3, 22, 917593,
    ↪tzinfo=tzutc()), 24.3],
    ['Prague', datetime.datetime(2022, 6, 7, 11, 3, 22, 917593, tzinfo=tzutc()),
    ↪ 25.3],
    ...
]

```

Serialization the query results to JSON via `to_json()`:

```

from influxdb_client.client.influxdb_client_async import InfluxDBClientAsync

async with InfluxDBClientAsync(url="http://localhost:8086", token="my-token",
    ↪org="my-org") as client:

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

    # Serialize to JSON
    output = tables.to_json(indent=5)
    print(output)

```

```

[
  {
    "_measurement": "mem",
    "_start": "2021-06-23T06:50:11.897825+00:00",
    "_stop": "2021-06-25T06:50:11.897825+00:00",
    "_time": "2020-02-27T16:20:00.897825+00:00",
    "region": "north",
    "_field": "usage",
    "_value": 15
  },
  {
    "_measurement": "mem",
    "_start": "2021-06-23T06:50:11.897825+00:00",
    "_stop": "2021-06-25T06:50:11.897825+00:00",
    "_time": "2020-02-27T16:20:01.897825+00:00",
    "region": "west",
    "_field": "usage",

```

(continues on next page)

(continued from previous page)

```

        "_value": 10
    },
    ...
]

```

**async query\_data\_frame**(*query: str, org=None, data\_frame\_index: Optional[List[str]] = None, params: Optional[dict] = None*)

Execute asynchronous Flux query and return `DataFrame`.

**Note:** If the query returns tables with differing schemas than the client generates a `DataFrame` for each of them.

#### Parameters

- **query** – the Flux query
- **org** (*str, Organization*) – specifies the organization for executing the query; Take the ID, Name or Organization. If not specified the default value from `InfluxDBClientAsync.org` is used.
- **data\_frame\_index** – the list of columns that are used as `DataFrame` index
- **params** – bind parameters

#### Returns

`DataFrame` or `List[DataFrame]`

**Warning:** For the optimal processing of the query results use the `pivot()` function which align results as a table.

```

from(bucket:"my-bucket")
  |> range(start: -5m, stop: now())
  |> filter(fn: (r) => r._measurement == "mem")
  |> pivot(rowKey:["_time"], columnKey: ["_field"], valueColumn: "_value")

```

#### For more info see:

- <https://docs.influxdata.com/resources/videos/pivots-in-flux/>
- <https://docs.influxdata.com/flux/latest/stdlib/universe/pivot/>
- <https://docs.influxdata.com/flux/latest/stdlib/influxdata/influxdb/schema/fieldsascols/>

**async query\_data\_frame\_stream**(*query: str, org=None, data\_frame\_index: Optional[List[str]] = None, params: Optional[dict] = None*)

Execute asynchronous Flux query and return stream of `DataFrame` as an `AsyncGenerator[DataFrame]`.

**Note:** If the query returns tables with differing schemas than the client generates a `DataFrame` for each of them.

#### Parameters

- **query** – the Flux query
- **org** (*str*, *Organization*) – specifies the organization for executing the query; Take the ID, Name or Organization. If not specified the default value from `InfluxDBClientAsync.org` is used.
- **data\_frame\_index** – the list of columns that are used as DataFrame index
- **params** – bind parameters

**Returns**

`AsyncGenerator[:class: `DataFrame`]`

**Warning:** For the optimal processing of the query results use the `pivot()` function which align results as a table.

```
from(bucket:"my-bucket")
  |> range(start: -5m, stop: now())
  |> filter(fn: (r) => r._measurement == "mem")
  |> pivot(rowKey:["_time"], columnKey: ["_field"], valueColumn: "_value")
```

**For more info see:**

- <https://docs.influxdata.com/resources/videos/pivots-in-flux/>
- <https://docs.influxdata.com/flux/latest/stdlib/universe/pivot/>
- <https://docs.influxdata.com/flux/latest/stdlib/influxdata/influxdb/schema/fieldsascols/>

**async query\_raw**(*query: str*, *org=None*, *dialect={'annotations': ['datatype', 'group', 'default'], 'comment\_prefix': '#', 'date\_time\_format': 'RFC3339', 'delimiter': ',', 'header': True}*, *params: Optional[dict] = None*)

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

**Parameters**

- **query** – a Flux query
- **org** (*str*, *Organization*) – specifies the organization for executing the query; Take the ID, Name or Organization. If not specified the default value from `InfluxDBClientAsync.org` is used.
- **dialect** – csv dialect format
- **params** – bind parameters

**Returns**

*str*

**async query\_stream**(*query: str*, *org=None*, *params: Optional[dict] = None*) → `AsyncGenerator[FluxRecord, None]`

Execute asynchronous Flux query and return stream of *FluxRecord* as an `AsyncGenerator[FluxRecord]`.

**Parameters**

- **query** – the Flux query
- **org** (*str*, *Organization*) – specifies the organization for executing the query; Take the ID, Name or Organization. If not specified the default value from `InfluxDBClientAsync.org` is used.

- **params** – bind parameters

**Returns**AsyncGenerator[*FluxRecord*]

### 3.3 WriteApiAsync

```
class influxdb_client.client.write_api_async.WriteApiAsync(influxdb_client, point_settings: ~influxdb_client.client.write_api.PointSettings = <influxdb_client.client.write_api.PointSettings object>)
```

Implementation for ‘/api/v2/write’ endpoint.

**Example:**

```
from influxdb_client_async import InfluxDBClientAsync

# Initialize async/await instance of Write API
async with InfluxDBClientAsync(url="http://localhost:8086", token="my-token",
    org="my-org") as client:
    write_api = client.write_api()
```

Initialize defaults.

**Parameters**

- **influxdb\_client** – with default settings (organization)
- **point\_settings** – settings to store default tags.

```
async write(bucket: str, org: Optional[str] = None, record: Optional[Union[str, Iterable[str], Point, Iterable[Point], dict, Iterable[dict], bytes, Iterable[bytes], NamedTuple, Iterable[NamedTuple], dataclass, Iterable[dataclass]]] = None, write_precision: WritePrecision = 'ns', **kwargs) → bool
```

Write time-series data into InfluxDB.

**Parameters**

- **bucket** (*str*) – specifies the destination bucket for writes (required)
- **org** (*str*, *Organization*) – specifies the destination organization for writes; take the ID, Name or Organization. If not specified the default value from `InfluxDBClientAsync.org` is used.
- **write\_precision** (*WritePrecision*) – specifies the precision for the unix timestamps within the body line-protocol. The precision specified on a `Point` has precedence and is used for write.
- **record** – `Point`, `Line Protocol`, `Dictionary`, `NamedTuple`, `Data Classes`, `Pandas DataFrame`

**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 data\_frame\_timestamp\_column**

name of DataFrame column which contains a timestamp. The column can be defined as a `str` value formatted as `2018-10-26`, `2018-10-26 12:00`, `2018-10-26 12:00:00-05:00` or other formats and types supported by `pandas.to_datetime` - DataFrame

**Key data\_frame\_timestamp\_timezone**

name of the timezone which is used for timestamp column - 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

**Returns**

True for successfully accepted data, otherwise raise an exception

**Example:**

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

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

**DataFrame:**

If the `data_frame_timestamp_column` is not specified the index of `Pandas DataFrame` is used as a timestamp for written data. The index can 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 = ...
```

(continues on next page)

(continued from previous page)

```
# 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')
```

## 3.4 DeleteApiAsync

**class** influxdb\_client.client.delete\_api\_async.DeleteApiAsync(*influxdb\_client*)

Async implementation for '/api/v2/delete' endpoint.

Initialize defaults.

**async delete**(*start: Union[str, datetime]*, *stop: Union[str, datetime]*, *predicate: str*, *bucket: str*, *org: Optional[Union[str, Organization]] = None*) → bool

Delete Time series data from InfluxDB.

### Parameters

- **start** (*str*, *datetime.datetime*) – start time
- **stop** (*str*, *datetime.datetime*) – stop time
- **predicate** (*str*) – predicate
- **bucket** (*str*) – bucket id or name from which data will be deleted
- **org** (*str*, *Organization*) – specifies the organization to delete data from. Take the ID, Name or Organization. If not specified the default value from InfluxDBClientAsync.org is used.

### Returns

True for successfully deleted data, otherwise raise an exception

## 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.

### 4.1 Before You Start

Please take a moment to review the following client docs:

- [User Guide, README.rst](#)
- [Examples](#)
- [API Reference](#)
- [CHANGELOG.md](#)

### 4.2 Content

- *Initializing Client*
- *Creating Database/Bucket*
- *Dropping Database/Bucket*
- **Writes**
  - *LineProtocol*
  - *Dictionary-style object*
  - *Structured data*
  - *Pandas DataFrame*
- *Querying*

## 4.3 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
```

## 4.4 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)
```

## 4.5 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)
```

## 4.6 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')
```

## 4.7 Writing Dictionary-style object

### influxdb-python

```
from influxdb import InfluxDBClient

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
        }
    }
]

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
            }
        }
    ]
```

(continues on next page)

(continued from previous page)

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

## 4.8 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')

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
```

(continues on next page)

(continued from previous page)

```

↪ 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"])

```

## 4.9 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')

```



## 4.10 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: -100000d)
|> filter(fn: (r) => r["_measurement"] == "cpu")
|> pivot(rowKey:["_time"], columnKey: ["_field"], valueColumn: "_value")
'''

    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](#).



## DEVELOPMENT

The following document covers how to develop the InfluxDB client library locally. Including how to run tests and build the docs.

- *tl;dr*
- *Getting Started*
- *Linting*
- *Testing*
  - *Code Coverage*
- *Documentation*

### 5.1 tl;dr

```
# from your forked repo, create and activate a virtualenv
python -m virtualenv venv
. venv/bin/activate
# install the library as editable with all dependencies
make install
# make edits
# run lint and tests
make lint test
```

### 5.2 Getting Started

1. Install Python

Most distributions include Python by default, so before going too far, try running `python --version` to see if it already exists. You may also have to specify `python3 --version`, for example, on Ubuntu.

2. Fork and clone the repo

The rest of this assumes you have cloned your fork of the upstream [client library](#) and are in the same directory of the forked repo.

3. Set up a virtual environment.

Python virtual environments let you install specific versioned dependencies in a contained manner. This way, you do not pollute or have conflicts on your system with different versions.

```
python -m virtualenv venv
. venv/bin/activate
```

Having a shell prompt change via [starship](#) or something similar is nice as it will let you know when and which virtual environment in you are in.

To exit the virtual environment, run `deactivate`.

#### 4. Install the client library

To install the local version of the client library run:

```
make install
```

This will install the library as editable with all dependencies. This includes all dependencies that are used for all possible features as well as testing requirements.

#### 5. Make changes and test

At this point, a user can make the required changes necessary and run any tests or scripts they have.

Before putting up a PR, the user should attempt to run the *lint* and *tests* locally. Lint will ensure the formatting of the code, while tests will run integration tests against an InfluxDB instance. For details on that set up see the next section.

```
make lint test
```

## 5.3 Linting

The library uses flake8 to do linting and can be run with:

```
make lint
```

## 5.4 Testing

The built-in tests assume that there is a running instance of InfluxDB 2.x up and running. This can be accomplished by running the `scripts/influxdb-restart.sh` script. It will launch an InfluxDB 2.x instance with Docker and make it available locally on port 8086.

Once InfluxDB is available, run all the tests with:

```
make test
```

### 5.4.1 Code Coverage

After running the tests, an HTML report of the tests is available in the `htmlcov` directory. Users can open `html/index.html` file in a browser and see a full report for code coverage across the whole project. Clicking on a specific file will show a line-by-line report of what lines were or were not covered.

## 5.5 Documentation

The docs are built using Sphinx. To build all the docs run:

```
make docs
```

This will build and produce a sample version of the web docs at `docs/_build/html/index.html`. From there the user can view the entire site and ensure changes are rendered correctly.

This repository contains the Python client library for use with InfluxDB 2.x and Flux. InfluxDB 3.x users should instead use the lightweight [v3 client library](#). InfluxDB 1.x users should use the [v1 client library](#).

For ease of migration and a consistent query and write experience, v2 users should consider using InfluxQL and the [v1 client library](#).

The API of the **influxdb-client-python** is not the backwards-compatible with the old one - **influxdb-python**.



## DOCUMENTATION

This section contains links to the client library documentation.

- [Product documentation](#), *Getting Started*
- [Examples](#)
- [API Reference](#)
- [Changelog](#)





## 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`\_**
- **`Logging`\_**

## INSTALLATION

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

**Python 3.7** 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`.

---

### 8.1 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
```

If your application uses `async/await` in Python you can install with the `async` extra:

```
$ pip install influxdb-client[async]
```

For more info see *How to use Asyncio*.

## 8.2 Setuptools

Install via [Setuptools](#).

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

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

## 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
```



## CLIENT CONFIGURATION

### 10.1 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
- `cert_file` - path to the certificate that will be used for mTLS authentication
- `cert_key_file` - path to the file contains private key for mTLS certificate
- `cert_key_password` - string or function which returns password for decrypting the mTLS private key
- `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")
```

```
[influx2]
url=http://localhost:8086
org=my-org
token=my-token
timeout=6000
verify_ssl=False
```

## 10.2 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\_CERT\_FILE - path to the certificate that will be used for mTLS authentication
- INFLUXDB\_V2\_CERT\_KEY\_FILE - path to the file contains private key for mTLS certificate
- INFLUXDB\_V2\_CERT\_KEY\_PASSWORD - string or function which returns password for decrypting the mTLS private key
- 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()
```

## 10.3 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, FluxCSVParse 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)
```

(continues on next page)



(continued from previous page)

```

    |> 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:

```

=====
Profiler: query
=====

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")

=====
Profiler: profiler/query
=====

result          : _profiler
table           : 0
_measurement     : profiler/query
TotalDuration   : 8924700
CompileDuration : 350900
QueueDuration   : 33800
PlanDuration    : 0
RequeueDuration : 0
ExecuteDuration : 8486500
Concurrency     : 0
MaxAllocated    : 2072
TotalAllocated  : 0
flux/query-plan :

digraph {
  ReadWindowAggregateByTime11
  // every = 1m, aggregates = [mean], createEmpty = true, timeColumn = "_stop"
  pivot8
  generated_yield

  ReadWindowAggregateByTime11 -> pivot8
  pivot8 -> generated_yield
}

influxdb/scanned-bytes: 0

```

(continues on next page)

(continued from previous page)

```

influxdb/scanned-values: 0

=====
Profiler: profiler/operator
=====
result          : _profiler
table           : 1
_measurement     : profiler/operator
Type            : *universe.pivotTransformation
Label           : pivot8
Count           : 3
MinDuration     : 32600
MaxDuration     : 126200
DurationSum     : 193400
MeanDuration    : 64466.666666666664

=====
Profiler: profiler/operator
=====
result          : _profiler
table           : 1
_measurement     : profiler/operator
Type            : *influxdb.readWindowAggregateSource
Label           : ReadWindowAggregateByTime11
Count           : 1
MinDuration     : 940500
MaxDuration     : 940500
DurationSum     : 940500
MeanDuration    : 940500.0

```

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:

```

Custom processing of profiler result: {'result': '_profiler', 'table': 0, '_measurement': 'profiler/query', 'TotalDuration': 18843792, 'CompileDuration': 1078666, 'QueueDuration': 93375, 'PlanDuration': 0, 'RequeueDuration': 0, 'ExecuteDuration': 17371000, 'Concurrency': 0, 'MaxAllocated': 448, 'TotalAllocated': 0, 'RuntimeErrors': None, 'flux/query-plan': 'digraph {\r\n  ReadRange2\r\n  generated_yield\r\n\r\n  ReadRange2 -> generated_yield\r\n}\r\n\r\n', 'influxdb/scanned-bytes': 0, 'influxdb/scanned-values': 0}
Custom processing of profiler result: {'result': '_profiler', 'table': 1, '_measurement': 'profiler/operator', 'Type': '*influxdb.readFilterSource', 'Label': 'ReadRange2', 'Count': 1, 'MinDuration': 3274084, 'MaxDuration': 3274084, 'DurationSum': 3274084, 'MeanDuration': 3274084.0}

```



## INDICES AND TABLES

- `genindex`
- `modindex`
- `search`



## A

add\_label() (influxdb\_client.TasksApi method), 55  
 add\_member() (influxdb\_client.TasksApi method), 55  
 add\_owner() (influxdb\_client.TasksApi method), 55  
 authorization\_id (influxdb\_client.domain.Task property), 58  
 authorizations\_api() (influxdb\_client.InfluxDBClient method), 26

## B

Bucket (class in influxdb\_client.domain), 47  
 buckets\_api() (influxdb\_client.InfluxDBClient method), 26  
 BucketsApi (class in influxdb\_client), 46  
 build() (influxdb\_client.client.influxdb\_client\_async.InfluxDBClientAsync method), 76  
 build() (influxdb\_client.InfluxDBClient method), 26

## C

cancel\_run() (influxdb\_client.TasksApi method), 55  
 clone\_label() (influxdb\_client.LabelsApi method), 49  
 clone\_task() (influxdb\_client.TasksApi method), 55  
 close() (influxdb\_client.client.influxdb\_client\_async.InfluxDBClientAsync method), 76  
 close() (influxdb\_client.InfluxDBClient method), 27  
 close() (influxdb\_client.WriteApi method), 41  
 create\_bucket() (influxdb\_client.BucketsApi method), 46  
 create\_label() (influxdb\_client.LabelsApi method), 49  
 create\_organization() (influxdb\_client.OrganizationsApi method), 50  
 create\_script() (influxdb\_client.InvokableScriptsApi method), 61  
 create\_task() (influxdb\_client.TasksApi method), 55  
 create\_task\_cron() (influxdb\_client.TasksApi method), 55  
 create\_task\_every() (influxdb\_client.TasksApi method), 55  
 create\_user() (influxdb\_client.UsersApi method), 53

created\_at (influxdb\_client.domain.Bucket property), 47  
 created\_at (influxdb\_client.domain.Organization property), 51  
 created\_at (influxdb\_client.domain.Script property), 66  
 created\_at (influxdb\_client.domain.Task property), 58  
 cron (influxdb\_client.domain.Task property), 58  
 CSVIterator (class in influxdb\_client.client.flux\_table), 39

## D

DateHelper (class in influxdb\_client.client.util.date\_utils), 70  
 default\_storage\_type (influxdb\_client.domain.Organization property), 51  
 delete() (influxdb\_client.client.delete\_api\_async.DeleteApiAsync method), 86  
 delete() (influxdb\_client.DeleteApi method), 69  
 delete\_api() (influxdb\_client.client.influxdb\_client\_async.InfluxDBClientAsync method), 76  
 delete\_api() (influxdb\_client.InfluxDBClient method), 27  
 delete\_bucket() (influxdb\_client.BucketsApi method), 46  
 delete\_label() (influxdb\_client.LabelsApi method), 50  
 delete\_label() (influxdb\_client.TasksApi method), 55  
 delete\_member() (influxdb\_client.TasksApi method), 55  
 delete\_organization() (influxdb\_client.OrganizationsApi method), 50  
 delete\_owner() (influxdb\_client.TasksApi method), 55  
 delete\_script() (influxdb\_client.InvokableScriptsApi method), 61  
 delete\_task() (influxdb\_client.TasksApi method), 55  
 delete\_user() (influxdb\_client.UsersApi method), 53  
 DeleteApi (class in influxdb\_client), 69  
 DeleteApiAsync (class in influxdb\_client.client.delete\_api\_async), 86

DeletePredicateRequest (class in influxdb\_client.domain), 69  
description (influxdb\_client.domain.Bucket property), 47  
description (influxdb\_client.domain.Organization property), 52  
description (influxdb\_client.domain.Script property), 66  
description (influxdb\_client.domain.ScriptCreateRequest property), 68  
description (influxdb\_client.domain.Task property), 58

## E

every (influxdb\_client.domain.Task property), 58

## F

field() (influxdb\_client.client.write.point.Point method), 42  
find\_bucket\_by\_id() (influxdb\_client.BucketsApi method), 46  
find\_bucket\_by\_name() (influxdb\_client.BucketsApi method), 46  
find\_buckets() (influxdb\_client.BucketsApi method), 46  
find\_label\_by\_id() (influxdb\_client.LabelsApi method), 50  
find\_label\_by\_org() (influxdb\_client.LabelsApi method), 50  
find\_labels() (influxdb\_client.LabelsApi method), 50  
find\_organization() (influxdb\_client.OrganizationsApi method), 51  
find\_organizations() (influxdb\_client.OrganizationsApi method), 51  
find\_scripts() (influxdb\_client.InvokableScriptsApi method), 61  
find\_task\_by\_id() (influxdb\_client.TasksApi method), 55  
find\_tasks() (influxdb\_client.TasksApi method), 55  
find\_tasks\_by\_user() (influxdb\_client.TasksApi method), 56  
find\_tasks\_iter() (influxdb\_client.TasksApi method), 56  
find\_users() (influxdb\_client.UsersApi method), 53  
flush() (influxdb\_client.WriteApi method), 41  
flux (influxdb\_client.domain.Task property), 59  
FluxRecord (class in influxdb\_client.client.flux\_table), 37  
FluxTable (class in influxdb\_client.client.flux\_table), 37  
from\_config\_file() (influxdb\_client.client.influxdb\_client\_async.InfluxDBClientAsync class method), 77

from\_config\_file() (influxdb\_client.InfluxDBClient class method), 27  
from\_dict() (influxdb\_client.client.write.point.Point static method), 43  
from\_env\_properties() (influxdb\_client.client.influxdb\_client\_async.InfluxDBClientAsync class method), 78  
from\_env\_properties() (influxdb\_client.InfluxDBClient class method), 29

## G

get\_field() (influxdb\_client.client.flux\_table.FluxRecord method), 38  
get\_group\_key() (influxdb\_client.client.flux\_table.FluxTable method), 37  
get\_labels() (influxdb\_client.TasksApi method), 56  
get\_logs() (influxdb\_client.TasksApi method), 56  
get\_measurement() (influxdb\_client.client.flux\_table.FluxRecord method), 38  
get\_members() (influxdb\_client.TasksApi method), 56  
get\_owners() (influxdb\_client.TasksApi method), 56  
get\_run() (influxdb\_client.TasksApi method), 56  
get\_run\_logs() (influxdb\_client.TasksApi method), 57  
get\_runs() (influxdb\_client.TasksApi method), 57  
get\_start() (influxdb\_client.client.flux\_table.FluxRecord method), 38  
get\_stop() (influxdb\_client.client.flux\_table.FluxRecord method), 38  
get\_time() (influxdb\_client.client.flux\_table.FluxRecord method), 38  
get\_value() (influxdb\_client.client.flux\_table.FluxRecord method), 38

## H

health() (influxdb\_client.InfluxDBClient method), 29

## I

id (influxdb\_client.domain.Bucket property), 47  
id (influxdb\_client.domain.Organization property), 52  
id (influxdb\_client.domain.Script property), 66  
id (influxdb\_client.domain.Task property), 59  
id (influxdb\_client.domain.User property), 54  
InfluxDBClient (class in influxdb\_client), 25  
InfluxDBClientAsync (class in influxdb\_client.client.influxdb\_client\_async), 75  
invokable\_scripts\_api() (influxdb\_client.InfluxDBClient method), 30  
InvokableScriptsApi (class in influxdb\_client), 61  
invoke\_script() (influxdb\_client.InvokableScriptsApi method), 62



invoke\_script\_csv() (influxdb\_client.InvokableScriptsApi method), 63  
 invoke\_script\_data\_frame() (influxdb\_client.InvokableScriptsApi method), 64  
 invoke\_script\_data\_frame\_stream() (influxdb\_client.InvokableScriptsApi method), 64  
 invoke\_script\_raw() (influxdb\_client.InvokableScriptsApi method), 65  
 invoke\_script\_stream() (influxdb\_client.InvokableScriptsApi method), 65  
 NS (influxdb\_client.domain.write\_precision.WritePrecision attribute), 45

## L

labels (influxdb\_client.domain.Bucket property), 48  
 labels (influxdb\_client.domain.Task property), 59  
 labels\_api() (influxdb\_client.InfluxDBClient method), 30  
 LabelsApi (class in influxdb\_client), 49  
 language (influxdb\_client.domain.Script property), 66  
 language (influxdb\_client.domain.ScriptCreateRequest property), 68  
 last\_run\_error (influxdb\_client.domain.Task property), 59  
 last\_run\_status (influxdb\_client.domain.Task property), 59  
 latest\_completed (influxdb\_client.domain.Task property), 59  
 links (influxdb\_client.domain.Bucket property), 48  
 links (influxdb\_client.domain.Organization property), 52  
 links (influxdb\_client.domain.Task property), 59

## M

me() (influxdb\_client.OrganizationsApi method), 51  
 me() (influxdb\_client.UsersApi method), 53  
 measurement() (influxdb\_client.client.write.point.Point static method), 44  
 MultiprocessingWriter (class in influxdb\_client.client.util.multiprocessing\_helper), 71

## N

name (influxdb\_client.domain.Bucket property), 48  
 name (influxdb\_client.domain.Organization property), 52  
 name (influxdb\_client.domain.Script property), 67  
 name (influxdb\_client.domain.ScriptCreateRequest property), 68  
 name (influxdb\_client.domain.Task property), 60  
 name (influxdb\_client.domain.User property), 54

## O

offset (influxdb\_client.domain.Task property), 60  
 org (influxdb\_client.domain.Task property), 60  
 org\_id (influxdb\_client.domain.Bucket property), 48  
 org\_id (influxdb\_client.domain.Script property), 67  
 org\_id (influxdb\_client.domain.Task property), 60  
 Organization (class in influxdb\_client.domain), 51  
 organizations\_api() (influxdb\_client.InfluxDBClient method), 30  
 OrganizationsApi (class in influxdb\_client), 50  
 owner\_id (influxdb\_client.domain.Task property), 60

## P

PandasDateTimeHelper (class in influxdb\_client.client.util.date\_utils\_pandas), 71  
 parse\_date() (influxdb\_client.client.util.date\_utils.DateHelper method), 70  
 parse\_date() (influxdb\_client.client.util.date\_utils\_pandas.PandasDateTimeHelper method), 71  
 ping() (influxdb\_client.client.influxdb\_client\_async.InfluxDBClientAsync method), 79  
 ping() (influxdb\_client.InfluxDBClient method), 30  
 Point (class in influxdb\_client.client.write.point), 42  
 predicate (influxdb\_client.domain.DeletePredicateRequest property), 69

## Q

query() (influxdb\_client.client.query\_api\_async.QueryApiAsync method), 80  
 query() (influxdb\_client.QueryApi method), 32  
 query\_api() (influxdb\_client.client.influxdb\_client\_async.InfluxDBClientAsync method), 79  
 query\_api() (influxdb\_client.InfluxDBClient method), 30  
 query\_csv() (influxdb\_client.QueryApi method), 34  
 query\_data\_frame() (influxdb\_client.client.query\_api\_async.QueryApiAsync method), 82  
 query\_data\_frame() (influxdb\_client.QueryApi method), 35  
 query\_data\_frame\_stream() (influxdb\_client.client.query\_api\_async.QueryApiAsync method), 82  
 query\_data\_frame\_stream() (influxdb\_client.QueryApi method), 36  
 query\_raw() (influxdb\_client.client.query\_api\_async.QueryApiAsync method), 83  
 query\_raw() (influxdb\_client.QueryApi method), 37  
 query\_stream() (influxdb\_client.client.query\_api\_async.QueryApiAsync method), 83

[query\\_stream\(\)](#) (*influxdb\_client.QueryApi method*), 37  
[QueryApi](#) (*class in influxdb\_client*), 32  
[QueryApiAsync](#) (*class in influxdb\_client.client.query\_api\_async*), 80

## R

[ready\(\)](#) (*influxdb\_client.InfluxDBClient method*), 30  
[retention\\_rules](#) (*influxdb\_client.domain.Bucket property*), 48  
[retry\\_run\(\)](#) (*influxdb\_client.TasksApi method*), 57  
[rp](#) (*influxdb\_client.domain.Bucket property*), 48  
[run\(\)](#) (*influxdb\_client.client.util.multiprocessing\_helper.MultiprocessingWriter method*), 72  
[run\\_manually\(\)](#) (*influxdb\_client.TasksApi method*), 57

## S

[schema\\_type](#) (*influxdb\_client.domain.Bucket property*), 48  
[Script](#) (*class in influxdb\_client.domain*), 66  
[script](#) (*influxdb\_client.domain.Script property*), 67  
[script](#) (*influxdb\_client.domain.ScriptCreateRequest property*), 68  
[ScriptCreateRequest](#) (*class in influxdb\_client.domain*), 67  
[set\\_str\\_rep\(\)](#) (*influxdb\_client.client.write.point.Point class method*), 44  
[start](#) (*influxdb\_client.domain.DeletePredicateRequest property*), 69  
[start\(\)](#) (*influxdb\_client.client.util.multiprocessing\_helper.MultiprocessingWriter method*), 73  
[status](#) (*influxdb\_client.domain.Organization property*), 52  
[status](#) (*influxdb\_client.domain.Task property*), 61  
[status](#) (*influxdb\_client.domain.User property*), 54  
[stop](#) (*influxdb\_client.domain.DeletePredicateRequest property*), 69

## T

[TableList](#) (*class in influxdb\_client.client.flux\_table*), 38  
[tag\(\)](#) (*influxdb\_client.client.write.point.Point method*), 44  
[Task](#) (*class in influxdb\_client.domain*), 57  
[tasks\\_api\(\)](#) (*influxdb\_client.InfluxDBClient method*), 30  
[TasksApi](#) (*class in influxdb\_client*), 55  
[terminate\(\)](#) (*influxdb\_client.client.util.multiprocessing\_helper.MultiprocessingWriter method*), 73  
[time\(\)](#) (*influxdb\_client.client.write.point.Point method*), 44  
[to\\_dict\(\)](#) (*influxdb\_client.domain.Bucket method*), 49  
[to\\_dict\(\)](#) (*influxdb\_client.domain.DeletePredicateRequest method*), 70  
[to\\_dict\(\)](#) (*influxdb\_client.domain.Organization method*), 52

[to\\_dict\(\)](#) (*influxdb\_client.domain.Script method*), 67  
[to\\_dict\(\)](#) (*influxdb\_client.domain.ScriptCreateRequest method*), 68  
[to\\_dict\(\)](#) (*influxdb\_client.domain.Task method*), 61  
[to\\_dict\(\)](#) (*influxdb\_client.domain.User method*), 54  
[to\\_dict\(\)](#) (*influxdb\_client.domain.write\_precision.WritePrecision method*), 45  
[to\\_json\(\)](#) (*influxdb\_client.client.flux\_table.TableList method*), 38  
[to\\_line\\_protocol\(\)](#) (*influxdb\_client.client.write.point.Point method*), 45  
[to\\_nanoseconds\(\)](#) (*influxdb\_client.client.util.date\_utils.DateHelper method*), 70  
[to\\_nanoseconds\(\)](#) (*influxdb\_client.client.util.date\_utils\_pandas.PandasDateTimeHelper method*), 71  
[to\\_str\(\)](#) (*influxdb\_client.domain.Bucket method*), 49  
[to\\_str\(\)](#) (*influxdb\_client.domain.DeletePredicateRequest method*), 70  
[to\\_str\(\)](#) (*influxdb\_client.domain.Organization method*), 52  
[to\\_str\(\)](#) (*influxdb\_client.domain.Script method*), 67  
[to\\_str\(\)](#) (*influxdb\_client.domain.ScriptCreateRequest method*), 68  
[to\\_str\(\)](#) (*influxdb\_client.domain.Task method*), 61  
[to\\_str\(\)](#) (*influxdb\_client.domain.User method*), 54  
[to\\_str\(\)](#) (*influxdb\_client.domain.write\_precision.WritePrecision method*), 45  
[to\\_utc\(\)](#) (*influxdb\_client.client.util.date\_utils.DateHelper method*), 70  
[to\\_values\(\)](#) (*influxdb\_client.client.flux\_table.CSVIterator method*), 39  
[to\\_values\(\)](#) (*influxdb\_client.client.flux\_table.TableList method*), 39  
[type](#) (*influxdb\_client.domain.Bucket property*), 49

## U

[update\\_bucket\(\)](#) (*influxdb\_client.BucketsApi method*), 47  
[update\\_label\(\)](#) (*influxdb\_client.LabelsApi method*), 50  
[update\\_organization\(\)](#) (*influxdb\_client.OrganizationsApi method*), 51  
[update\\_password\(\)](#) (*influxdb\_client.UsersApi method*), 53  
[update\\_script\(\)](#) (*influxdb\_client.InvokableScriptsApi method*), 66  
[update\\_task\(\)](#) (*influxdb\_client.TasksApi method*), 57  
[update\\_task\\_request\(\)](#) (*influxdb\_client.TasksApi method*), 57  
[update\\_user\(\)](#) (*influxdb\_client.UsersApi method*), 54

[updated\\_at \(influxdb\\_client.domain.Bucket property\), 49](#)  
[updated\\_at \(influxdb\\_client.domain.Organization property\), 52](#)  
[updated\\_at \(influxdb\\_client.domain.Script property\), 67](#)  
[updated\\_at \(influxdb\\_client.domain.Task property\), 61](#)  
[url \(influxdb\\_client.domain.Script property\), 67](#)  
[User \(class in influxdb\\_client.domain\), 54](#)  
[users\\_api\(\) \(influxdb\\_client.InfluxDBClient method\), 30](#)  
[UsersApi \(class in influxdb\\_client\), 53](#)

## V

[version\(\) \(influxdb\\_client.client.influxdb\\_client\\_async.InfluxDBClientAsync method\), 79](#)  
[version\(\) \(influxdb\\_client.InfluxDBClient method\), 30](#)

## W

[write\(\) \(influxdb\\_client.client.util.multiprocessing\\_helper.MultiprocessingWriter method\), 73](#)  
[write\(\) \(influxdb\\_client.client.write\\_api\\_async.WriteApiAsync method\), 84](#)  
[write\(\) \(influxdb\\_client.WriteApi method\), 41](#)  
[write\\_api\(\) \(influxdb\\_client.client.influxdb\\_client\\_async.InfluxDBClientAsync method\), 80](#)  
[write\\_api\(\) \(influxdb\\_client.InfluxDBClient method\), 31](#)  
[write\\_precision \(influxdb\\_client.client.write.point.Point property\), 45](#)  
[WriteApi \(class in influxdb\\_client\), 40](#)  
[WriteApiAsync \(class in influxdb\\_client.client.write\\_api\\_async\), 84](#)  
[WritePrecision \(class in influxdb\\_client.domain.write\\_precision\), 45](#)