datastream Documentation

Release 0.4.3

wlan slovenija

Sep 27, 2017

Contents

Datastream API is one of the projects of wlan slovenija open wireless network. It is a Python API for time-series data which abstracts the database which is used to store the data, providing a powerful and unified API. It provides an easy way to insert time-series datapoints and automatically downsample them into multiple levels of granularity for efficient querying time-series data at various time scales.

CHAPTER 1

Contents

Installation

Inside a virtualenv, using pip simply by doing:

```
pip install datastream
```

Or install from source directly.

Usage

Datastream API provides a Python interface which you can initialize with MongoDB backend by:

```
import datastream
from datastream.backends import mongodb

stream = datastream.Datastream(mongodb.Backend('database_name'))
```

MongoDB backend accepts some additional connection settings, if this is needed.

After that you can create new streams, insert datapoints into them and query streams. See *API reference* for more information.

Tags

Each stream can have arbitrary JSON-serializable metadata associated to it through arbitrary tags. You can then query streams by using those tags. Some tags are reserved to not conflict with stream settings and some tags are used by higher-level packages like django-datastream. Although tags can be complex values, simple values like strings or simple dicts are preferred.

Types

Datastream API supports various types for values stored as datapoints. Types influence how downsampling is done. Currently supported types are:

- `numeric` – each datapoint value is a number
- `nominal` – each datapoint value is a an arbitrary value, but most often a simple label
- `graph` – each datapoint value is a graph

Numeric values can be integers, floats, `decimal.Decimal`, or any other instance of `numbers.Number`. Alternatively, one can append an already downsampled value in the same format and with all values downsampled values for a given stream have. This is useful when the source of their values already provides information from multiple samples. For example, pinging over the Internet sends multiple packets and then returns min, max, mean times. By storing directly min, max, and mean values, no information is lost and can be reused by Datastream API.

Nominal values (also known as qualitative) can be any JSON-serializable arbitrary value, but most often they are a simple label. Values are stored as-is in the database so repeating the same huge value multiple times will be stored multiple times. If values will be repeating it is better to instead store only some small keys representing them. Nominal values do not have a defined order between them.

Graph values are stored as dicts in the format:

```
{
    "v": [
        {"i": "foo"},
        {"i": "bar"}
    ],
    "e": [
        {"f": "foo", "t": "bar"}
    ]
}
```

It contains a list of vertices `v` where each vertex element contains its ID `i`. IDs can be of arbitrary type. Vertices can contain additional fields which are ignored, but might be used by downsamplers. List of edges `e` contains edges from vertex with ID equal to `f`, to vertex with ID equal to `t`. Additional fields are ignored, but might be used by downsamplers as well.

Downsampling

Datastream API automatically downsample datapoints to lower granularity levels. Highest supported resolution for datapoints is a second, and then Datastream API will downsample them. If you know that you will insert datapoints at lower granularity levels (for example, only every 5 minutes), you can specify that so that Datastream API can optimize.

Downsampling happens both for the datapoint value and the datapoint timestamp. It takes a list of datapoints for a timespan at a higher granularity level and creates a downsampled value and downsampled timestamp for a datapoint at a lower granularity level. You can configure what exactly this downsampled datapoint contains. You can for example configure that it contains a mean, minimum and maximum of all values from a timespan. Same for the timestamp, for example, you can configure that timestamp for the datapoint contains first, last and mean timestamps of all datapoints from a timespan.

All downsampling timespans for all streams are equal and rounded at reasonable boundaries (for example, hour granularity starts and ends at full hour).

Derived Streams

Datastream API supports derived streams. Streams which are automatically generated from other streams as new datapoints are appended to those streams. For example, you can create a stream which computes derivative of another stream. Or sums multiple streams together.

Django HTTP Interface

We provide a Django HTTP RESTful interface through django-datastream package. You can use it directly in your Django application, or check its source code to learn more how to integrate Datastream API into your application.

Reference

API

class `datastream.api.`**`Datastream`**(*backend*)

Initializes the Datastream API.

Parameters **`backend`** – Backend instance

`append`(*stream_id*, *value*, *timestamp=None*, *check_timestamp=True*)

Appends a datapoint into the datastream.

Parameters

- **`stream_id`** – Stream identifier
- **`value`** – Datapoint value
- **`timestamp`** – Datapoint timestamp, must be equal or larger (newer) than the latest one, monotonically increasing (optional)
- **`check_timestamp`** – Check if timestamp is equal or larger (newer) than the latest one (default: true)

Returns A dictionary containing *stream_id*, *granularity*, and *datapoint*

`backprocess_streams`(*query_tags=None*)

Requests the backend to backprocess any derived streams.

Parameters **`query_tags`** – Tags that should be matched to streams

`clear_tags`(*stream_id*)

Removes (clears) all non-readonly stream tags.

Care should be taken that some tags are set immediately afterwards which uniquely identify a stream to be able to query the stream, in for example, *ensure_stream*.

Parameters **`stream_id`** – Stream identifier

`delete_streams`(*query_tags=None*)

Deletes datapoints for all streams matching the specified query tags. If no query tags are specified, all datastream-related data is deleted from the backend.

Parameters **`query_tags`** – Tags that should be matched to streams

`downsample_streams`(*query_tags=None*, *until=None*, *return_datapoints=False*)

Requests the backend to downsample all streams matching the specified query tags. Once a time range has been downsampled, new datapoints cannot be added to it anymore.

Parameters

- **query_tags** – Tags that should be matched to streams
- **until** – Timestamp until which to downsample, not including datapoints at a timestamp (optional, otherwise all until the current time)
- **return_datapoints** – Should newly downsampled datapoints be returned, this can potentially create a huge temporary list and memory consumption when downsampling many streams and datapoints

Returns A list of dictionaries containing *stream_id*, *granularity*, and *datapoint* for each datapoint created while downsampling, if *return_datapoints* was set

ensure_stream(*query_tags*, *tags*, *value_downsamplers*, *highest_granularity*, *derive_from=None*, *derive_op=None*, *derive_args=None*, *value_type=None*, *value_type_options=None*)

Ensures that a specified stream exists.

Parameters

- **query_tags** – A dictionary of tags which uniquely identify a stream
- **tags** – A dictionary of tags that should be used (together with *query_tags*) to create a stream when it doesn't yet exist
- **value_downsamplers** – A set of names of value downsampler functions for this stream
- **highest_granularity** – Predicted highest granularity of the data the stream will store, may be used to optimize data storage
- **derive_from** – Create a derivate stream
- **derive_op** – Derivation operation
- **derive_args** – Derivation operation arguments
- **value_type** – Optional value type (defaults to *numeric*)
- **value_type_options** – Options specific to the value type

Returns A stream identifier

find_streams(*query_tags=None*)

Finds all streams matching the specified query tags.

Parameters **query_tags** – Tags that should be matched to streams

Returns A *Streams* iterator over matched stream descriptors

get_data(*stream_id*, *granularity*, *start=None*, *end=None*, *start_exclusive=None*, *end_exclusive=None*, *reverse=False*, *value_downsamplers=None*, *time_downsamplers=None*)

Retrieves data from a certain time range and of a certain granularity.

Parameters

- **stream_id** – Stream identifier
- **granularity** – Wanted granularity
- **start** – Time range start, including the start
- **end** – Time range end, excluding the end (optional)
- **start_exclusive** – Time range start, excluding the start
- **end_exclusive** – Time range end, excluding the end (optional)

- **reverse** – Should datapoints be returned in oldest to newest order (false), or in reverse (true)
- **value_downsamplers** – The list of downsamplers to limit datapoint values to (optional)
- **time_downsamplers** – The list of downsamplers to limit timestamp values to (optional)

Returns A *Datapoints* iterator over datapoints

get_tags(*stream_id*)
Returns the tags for the specified stream.

Parameters **stream_id** – Stream identifier

Returns A dictionary of tags for the stream

remove_tag(*stream_id*, *tag*)
Removes a stream tag.

Parameters

- **stream_id** – Stream identifier
- **tag** – Dictionary describing the tag(s) to remove (values are ignored)

update_tags(*stream_id*, *tags*)
Updates stream tags with new tags, overriding existing ones.

Parameters

- **stream_id** – Stream identifier
- **tags** – A dictionary of new tags

Backends

API operations are implemented in backends, which are responsible for storing datapoints, performing downsampling, deriving streams, and executing queries.

class `datastream.backends.mongodb.`**Backend**(*database_name*, ***connection_settings*)
Initializes the MongoDB backend.

Parameters

- **database_name** – MongoDB database name
- **connection_settings** – Extra connection settings as defined for *mongoengine.register_connection*

Implementation Details

Streams are stored in the `streams` collection, datapoints are stored in the `datapoints.<granularity>` collections, where `<granularity>` is one of the possible granularity levels.

When performing downsampling, we have to differentiate between two timestamps:

- Datapoint timestamp is the timestamp of the datapoint that has been inserted for a given granularity level. On the highest granularity level it is always second precision. On lower granularity levels it is a dictionary of multiple values, depending on time downsamplers settings for a given stream.

- Internal datapoint timestamp (stored in datapoint's `_id`) is based on a timespan for the given granularity level. For example, if a datapoint was inserted at 31-07-2012 12:23:52, then the downsampled internal timestamp for the timespan this datapoint is in for hour granularity would be 31-07-2012 12:00:00 and for month granularity would be 01-07-2012 00:00:00.

Based on `highest_granularity` value, appended datapoints are stored in the collection configured by `highest_granularity` and only lower granularity values are downsampled. Requests for granularity higher than `highest_granularity` simply return values from `highest_granularity` collection. `highest_granularity` is just an optimization to not store unnecessary datapoints for granularity levels which would have at most one datapoint for their granularity timespans.

Value Downsamplers

mean (*key: m*)
Average of all datapoints.

sum (*key: s*)
Sum of all datapoints.

min (*key: l, for lower*)
Minimum value of all dataponts.

max (*key: u, for upper*)
Maximum value of all datapoints.

sum_squares (*key: q*)
Sum of squares of all datapoints.

std_dev (*key: d*)
Standard deviation of all datapoints.

count (*key: c*)
Number of all datapoints.

most_often (*key: o, for often*)
The most often occurring value of all datapoints.

least_often (*key: r, for rare*)
The least often occurring value of all datapoints.

frequencies (*key: f*)
For each value number of occurrences in all datapoints.

Time Downsamplers

mean (*key: m*)
Average of all timestamps.

first (*key: a, is the first in the alphabet*)
The first timestamp of all datapoints.

last (*key: z, is the last in the alphabet*)
The last timestamp of all datapoints.

Derive Operators

sum(*src_streams*, *dst_stream*)
Sum of multiple streams.

derivative(*src_stream*, *dst_stream*)
Derivative of a stream.

counter_reset(*src_stream*, *dst_stream*)
Generates a counter reset stream.

counter_derivative([*{'name': 'reset', 'stream': reset_stream_id}*, *{'stream': data_stream_id}*], *dst_stream*, *max_value=None*)
Derivative of a monotonically increasing counter stream.

Exceptions

exception datastream.exceptions.**DatastreamException**(**args*, ***kwargs*)
The base class for all datastream API exceptions.

exception datastream.exceptions.**StreamNotFound**(**args*, ***kwargs*)
Raised when stream queried for is not found.

exception datastream.exceptions.**MultipleStreamsReturned**(**args*, ***kwargs*)
Raised when multiple streams found when queried for operations which operate on only one stream, like *ensure_stream()*. Specify more specific query tags.

exception datastream.exceptions.**InconsistentStreamConfiguration**(**args*, ***kwargs*)
Raised when stream configuration passed to *ensure_stream()* is inconsistent and/or conflicting.

exception datastream.exceptions.**OutstandingDependenciesError**(**args*, ***kwargs*)
Raised when stream cannot be deleted because it is a dependency for another stream.

exception datastream.exceptions.**UnsupportedDownsampler**(**args*, ***kwargs*)
Raised when downsampler requested is unsupported.

exception datastream.exceptions.**UnsupportedGranularity**(**args*, ***kwargs*)
Raised when granularity level requested is unsupported.

exception datastream.exceptions.**UnsupportedDeriveOperator**(**args*, ***kwargs*)
Raised when derive operator requested is unsupported.

exception datastream.exceptions.**UnsupportedValueType**(**args*, ***kwargs*)
Raised when value type requested is unsupported.

exception datastream.exceptions.**ReservedTagNameError**(**args*, ***kwargs*)
Raised when updating tags with a reserved tag name.

exception datastream.exceptions.**InvalidTimestamp**(**args*, ***kwargs*)
Raised when an invalid timestamp was provided.

exception datastream.exceptions.**IncompatibleGranularities**(**args*, ***kwargs*)
Raised when derived stream's granularity is incompatible with source stream's granularity.

exception datastream.exceptions.**IncompatibleTypes**(**args*, ***kwargs*)
Raised when derived stream's value type is incompatible with source stream's value type.

exception datastream.exceptions.**AppendToDerivedStreamNotAllowed**(**args*, ***kwargs*)
Raised when attempting to append to a derived stream.

exception datastream.exceptions.**InvalidOperatorArguments**(**args*, ***kwargs*)
Raised when derive operators received invalid arguments.

exception datastream.exceptions.**LockExpiredMidMaintenance**(**args*, ***kwargs*)
Raised when a maintenance lock expires inside a maintenance operation.

exception datastream.exceptions.**StreamAppendContended**(**args*, ***kwargs*)
Raised when too many processes are trying to append to the same stream.

exception datastream.exceptions.**DatastreamWarning**(**args*, ***kwargs*)
The base class for all datastream API runtime warnings.

exception datastream.exceptions.**InvalidValueWarning**(**args*, ***kwargs*)
Warning used when an invalid value is encountered.

exception datastream.exceptions.**InternalInconsistencyWarning**(**args*, ***kwargs*)
Warning used when an internal inconsistency is detected.

exception datastream.exceptions.**DownsampleConsistencyNotGuaranteed**(**args*, ***kwargs*)
Warning used when consistency of downsampled values with original datapoints is no longer guaranteed due to some condition. Reseting downsample state and redoing downsampling could be necessary.

CHAPTER 2

Source Code, Issue Tracker and Mailing List

For development GitHub is used, so source code and issue tracker is found there. If you have any questions or if you want to discuss the project, use development mailing list.

CHAPTER 3

Indices and Tables

- genindex
- search

Python Module Index

d

Index