python_moztelemetry Documentation

Release 0.3.9.13

Mozilla Firefox Data Platform

May 15, 2017

Contents

A simple library to fetch and analyze data collected by the Mozilla Telemetry service. Objects collected by Telemetry are called `pings`. A ping has a number of properties (aka `dimensions`) and a payload. A session of Telemetry data analysis/manipulation typically starts with a query that filters the objects by one or more dimensions. This query can be expressed using either an orm-like api, *Dataset* or a simple function, *get_pings() (deprecated)*.

CHAPTER 1

API

Dataset

class moztelemetry.dataset.**Dataset**(*bucket*, *schema*, *store=None*, *prefix=None*, *clauses=None*, *selection=None*)

Represents a collection of objects on S3.

A Dataset can have zero, one or many filters, which are refined using the *where* method. The result of refining a Dataset is a Dataset itself, so it's possible to chain multiple *where* clauses together.

The actual data retrieval is triggered by the *records* method, which returns a Spark RDD containing the list of records retrieved. To call *records* a SparkContext object must be provided.

Usage example:

```
bucket = 'test-bucket'
schema = ['submissionDate', 'docType', 'platform']

records = Dataset(bucket, schema) \
    .select(
        'clientId',
        os_name='environment.system.os.name',
        first_paint='payload.simpleMeasurements.firstPaint',
        // Take the first 2 stacks for each thread hang.
        stack_list='payload.threadHangStats[].hangs[].stack[0:2]'
    ).where(
        docType='main',
        appUpdateChannel='nightly',
        submissionDate=lambda x: x.startswith('201607'),
    ).records(sc)
```

For convenience Dataset objects can be created using the factory method *from_source*, that takes a source name (e.g. 'telemetry') and returns a new Dataset instance. The instance created will we aware of the list of dimensions, available on its *schema* attribute for inspection.

static from_source(*source_name*)

Create a Dataset configured for the given source_name

This is particularly convenient when the user doesn't know the list of dimensions or the bucket name, but only the source name.

Usage example:

```
        records = Dataset.from_source('telemetry').where(
            docType='main',
```

```
            submissionDate='20160701',
            appUpdateChannel='nightly'
        )
```

records(*sc*, *limit=None*, *sample=1*, *seed=42*, *decode=None*, *summaries=None*)
Retrieve the elements of a Dataset

Parameters

- **sc** – a SparkContext object
- **limit** – maximum number of objects to retrieve
- **decode** – an optional transformation to apply to the objects retrieved
- **sample** – percentage of results to return. Useful to return a sample of the dataset. This parameter is ignored when *limit* is set.
- **seed** – initialize internal state of the random number generator (42 by default). This is used to make the dataset sampling reproducible. It can be set to None to obtain different samples.
- **summaries** – an iterable containing a summary for each item in the dataset. If None, it will computed calling the summaries dataset.

Returns a Spark rdd containing the elements retrieved

select(**properties*, ***aliased_properties*)
Specify which properties of the dataset must be returned

Property extraction is based on JMESPath expressions. This method returns a new Dataset narrowed down by the given selection.

Parameters

- **properties** – JMESPath to use for the property extraction. The JMESPath string will be used as a key in the output dictionary.
- **aliased_properties** – Same as properties, but the output dictionary will contain the parameter name instead of the JMESPath string.

summaries(*sc*, *limit=None*)
Summary of the files contained in the current dataset

Every item in the summary is a dict containing a key name and the corresponding size of the key item in bytes, e.g.:: {'key': 'full/path/to/my/key', 'size': 200}

Parameters **limit** – Max number of objects to retrieve

Returns An iterable of summaries

where(***kwargs*)
Return a new Dataset refined using the given condition

Parameters **kwargs** – a map of *dimension* => *condition* to filter the elements of the dataset. *condition* can either be an exact value or a callable returning a boolean value.

get_pings() (deprecated)

moztelemetry.spark.**get_pings**(*sc*, *app=None*, *build_id=None*, *channel=None*, *doc_type='saved_session'*, *fraction=1.0*, *schema=None*, *source_name='telemetry'*, *source_version='4'*, *submission_date=None*, *version=None*)

Returns a RDD of Telemetry submissions for a given filtering criteria.

Parameters

- **sc** – an instance of SparkContext
- **app** – an application name, e.g.: "Firefox"
- **channel** – a channel name, e.g.: "nightly"
- **version** – the application version, e.g.: "40.0a1"
- **build_id** – a build_id or a range of build_ids, e.g.: "20150601000000" or ("20150601000000", "20150610999999")
- **submission_date** – a submission date or a range of submission dates, e.g: "20150601" or ("20150601", "20150610")
- **source_name** – source name, set to "telemetry" by default
- **source_version** – source version, set to "4" by default
- **doc_type** – ping type, set to "saved_session" by default
- **schema** – (deprecated) version of the schema to use
- **fraction** – the fraction of pings to return, set to 1.0 by default

Using Spark RDDs

Both `Dataset` and `get_pings` return the data as a Spark RDD. Users can then use the RDD api to further shape or transform the dataset.

Experimental APIs

moztelemetry.zeppelin.**show**(*fig*, *width=600*)

Renders a Matplotlib figure in Zeppelin.

Parameters

- **fig** – a Matplotlib figure
- **width** – the width in pixel of the rendered figure, defaults to 600

Usage example:

```
import matplotlib.pyplot as plt
from moztelemetry.zeppelin import show

fig = plt.figure()
plt.plot([1, 2, 3])
show(fig)
```

class moztelemetry.hbase.**HBaseMainSummaryView**(*hostname=None*)

The access gateway to the HBase main summary view

The gateway allows to retrieve the history of pings for a *small* set of client ids. The retrieval can optionally be limited to a time period of activity for said clients.

Usage example:

```
view = HBaseMainSummaryView()

for client_id, pings in view.get(sc, ["00000000-0000-0000-0000-000000000000"], limit=10).collect
        print client_id
        for ping in pings:
            print ping["subsession_start_date"]

for client_id, pings in view.get_range(sc, ["00000000-0000-0000-0000-000000000000"],
        range_start=date(2016, 12, 1), range_end=date(2016, 12, 2)).collect():
        print client_id
        for ping in pings:
            print ping["subsession_start_date"]
```

Note that retrieving the whole ping is not only slower, but also not needed for most analyses. Please try to collect back to the driver only the data you really need.

Fast retrieval example:

```
for client_id, pings in view.get(sc, ["00000000-0000-0000-0000-000000000000"], limit=10)
    print client_id
    for ping in pings:
        print ping
```

get(*sc*, *client_ids*, *limit=None*, *parallelism=None*, *reverse=False*)

Return RDD[client_id, [ping1, ..., pingK]]

The list of pings is sorted by activity date.

Parameters

- **sc** – a SparkContext
- **client_ids** – the client ids represented as UUIDs
- **limit** – the maximum number of pings to return per client id
- **parallelism** – the number of partitions of the resulting RDD
- **reverse** – whether to return pings in reverse chronological order, defaults to False

get_range(*sc*, *client_ids*, *range_start*, *range_end*, *limit=None*, *parallelism=None*, *reverse=False*)

Return RDD[client_id, [ping1, ..., pingK]] where pings are limited to a given activity period.

The list of pings is sorted by activity date.

Parameters

- **sc** – a SparkContext
- **client_ids** – the client ids represented as UUIDs
- **range_start** – the beginning of the time period represented as a datetime.date instance
- **range_end** – the end of the time period (inclusive) represented as a datetime.date instance
- **limit** – the maximum number of pings to return per client id

- **parallelism** – the number of partitions of the resulting RDD
- **reverse** – whether to return pings in reverse chronological order, defaults to False

CHAPTER 2

Indices and tables

- genindex
- modindex
- search

m

D

F

G

H

M

R

S

W