Data Access

The data access is a Python tool used to access data associated to the various scenario instances of an OpenBACH platform. Optionally, the data access can be used to modify, create, or delete such data.

Please refer to the Introduction to OpenBACH-extra and OpenBACH API page to see how to install and set up your platform to use the API described in this page.

Description

The data access contains tools to interact with and parse responses from the InfluxDB and ElasticSearch APIs used as part of the OpenBACH platform. These tools are hosted respectively in the `data_access.influxdb_tools` and `data_access.elasticsearch_tools` modules. Both modules are tied together in the `data_access.collector` module which is considered the public interface of the data access. The `data_access` package itself only export content from the collector module as its main class, for instance.

Each of these modules will return `Scenario` objects from the `data_access.result_data` module when asked for data about one or several scenario instances.

The `data_access.async_collector` module simply wraps the methods of the `data_access.collector.CollectorConnection` class into an asynchronous executor call.

Lastly the `data_access.post_processing` module is an extension to the `data_access.influxdb_tools` module that helps building jobs whose aim are to plot data from job instances data.

Collector

Common methods

`data_access.collector.CollectorConnection`, `data_access.influxdb_tools.InfluxDBConnection` and `data_access.elasticsearch_tools.ElasticSearchConnection` all share some common methods. They query the associated database and return a set containing the requested data. As you may imagine, `InfluxDBConnection` will query the InfluxDB database to return its information, `ElasticSearchConnection` will query the ElasticSearch database to return its information and `CollectorConnection` will return the union of the sets returned by the other two classes.

On these methods, parameters are shared by both databases except for:

- *suffix*: as it pertain to statistics emitted, it is only available in InfluxDB;
- *condition*: restrict InfluxDB query with a custom `WHERE` clause, see below for informations on how to build conditions;
- *timestamps*: a number or a pair of numbers, used to select a moment or an interval in

ElasticSearch. A few methods share it to InfluxDB as well to create a query which includes a clause `WHERE time=…` or a clause `WHERE (time >= …) AND (time ⇐ …)`.

Other possible parameters are used to restrict the data gathered:

- *job_name* (or *job* in databases classes) restricts to data generated by jobs of this name;
- *scenario_instance_id* (or *scenario* in databases classes) restricts to data generated by the scenario having this instance id;
- *agent_name* (or *agent* in databases classes) restricts to data generated by jobs running on this agent;
- *job_instance_id* (or *job_instance* in databases classes) restricts to data generated by the job having this instance id;

The following common methods are found on all three classes and return a set:

- *agent_names*: used to retrieve the names of the agents that generated data for the given constraints. Optional parameters:
  - *job_name*
  - *scenario_instance_id*
  - *job_instance_id*
  - *suffix*
  - *timestamps*
- *job_names*: used to retrieve the names of the jobs that generated data for the given constraints. Optional parameters:
  - *scenario_instance_id*
  - *agent_name*
  - *job_instance_id*
  - *suffix*
  - *timestamps*
- *job_instance_ids*: used to retrieve the IDs of job instances that generated data for the given constraints. Optional parameters:
  - *job_name*
  - *scenario_instance_id*
  - *agent_name*
  - *suffix*
  - *timestamps*
- *scenario_instance_ids*: used to retrieve the IDs of scenario instances that generated data for the given constraints. Optional parameters:
  - *job_name*
  - *agent_name*
  - *job_instance_id*
  - *suffix*
  - *timestamps*
- *timestamps*: used to retrieve the timestamps at which data has been stored in the databases. Optional parameters:
  - *job_name*
  - *scenario_instance_id*
  - *agent_name*
  - *job_instance_id*
  - *suffix*
  - *condition*

    - *only_bounds*: boolean indicating whether to return a sorted list of the timestamps or only a pair of the interval bounds; default to `True` (only a pair)
- *suffixes*: used to retrieve the suffixes in InfluxDB that are associated to the data generated for the given constraints. Optional parameters:
    - *job_name*
    - *scenario_instance_id*
    - *agent_name*
    - *job_instance_id*
- *remove_statistics*: used to delete data that matches the given constraints in the databases. Does not return anything. Dispatches to *remove_logs* in `ElasticSearchConnection`. Optional parameters:
    - *job_name*
    - *scenario_instance_id*
    - *agent_name*
    - *job_instance_id*
    - *suffix*
    - *condition*
    - *timestamps*: is ANDed with *condition* (if one is provided) when querying InfluxDB

The following method uses `data_access.result_data.Scenario` objects rather than mere identifiers:

- *scenarios*: generate `Scenario` instances based off the data matching the given constraints. This is the only method that will give you access to the actual data generated by your OpenBACH scenarios. Dispatches to *statistics* in `InfluxDBConnection` and *logs* in `ElasticSearchConnection`. Optional parameters:
    - *job_name*
    - *scenario_instance_id*
    - *agent_name*
    - *job_instance_id*
    - *suffix*
    - *fields*: a list of field (statistic name) that will be turned into the SELECT clause when querying InfluxDB, defaults to SELECT * if not provided. Can also be a string to fine tune the select clause, in which case it will be used as-is (`"SELECT {}".format(fields)`); note that, in this case, you most likely want to include the OpenBACH tags manually or the `Scenario` instance would be impossible to construct and return.
    - *condition*
    - *timestamps*: is ANDed with *condition* (if one is provided) when querying InfluxDB
- *import_scenario*: takes a `Scenario` instance as parameter and dumps its data into both databases. Dispatches to *import_job* in both databases classes. Does not return anything.
- *orphans*: retrieve data that is not associated with any OpenBACH scenario and return a pair comprising a `Log` and a `Scenario` instance. Optional parameters:
    - *condition*
    - *timestamps*: is ANDed with *condition* (if one is provided) when querying InfluxDB
- The returned values have specific meanings:
    - `ElasticSearchConnection.orphans` return a `Log` instance (first element of the pair) that holds ElasticSearch data that were not emitted using the collect-agent API. This is most likely warnings and errors generated by the controller or the agents themselves, out of a job execution;
    - `InfluxDBConnection.orphans` return a `Scenario` instance (second element of the

pair) that has no instance ID and whose `Jobs` consist of measurements found without the proper OpenBACH tags.

InfluxDB Utilities

On top of the common methods, `InfluxDBConnection` provide a few more specific methods:

- *get_field_keys*: return a dictionary of all the fields (statistic names) for each measurement (job names).
- *origin*: retrieve the first timestamp in InfluxDB that corresponds to the given constraints. Optional parameters:
  - *job*
  - *scenario*
  - *agent*
  - *job_instance*
  - *suffix*
  - *condition*
- *raw_statistics*: accepts the same parameters than *statistics* but return "raw" results instead of a `Scenario` instance. Raw results are an iterable of pairs *measurement_name, dictionary of a line of the measurement*.
- *sql_query*: base method to send a GET request to InfluxDB; accepts the raw SQL query as parameter and returns the JSON data that InfluxDB sent back.
- *data_write*: base method to send a POST request to InfluxDB; accepts the raw request body string as parameter and returns nothing.

The `data_access.influxdb_tools` module also provide utility functions:

- *tags_to_condition*: create a suitable `Condition` instance from some OpenBACH tags, ready to be used in one of the following `*_query` function.
- *select_query*: create an InfluxDB query string to fetch data from one or all measurements. Optionally accepts the job name (measurement), the field names (statistics) and a restricting condition.
- *measurement_query*: create an InfluxDB query string to show the measurements (job names) that holds data suitable for the given optional condition.
- *delete_query*: create an InfluxDB query string to remove data from the InfluxDB database.
- *tag_query*: create an InfluxDB query string to show the values associated to a given tag. Optionally accepts a job name (measurement) and a restricting condition.
- *parse_influx*: accepts the raw JSON from an InfluxDB SQL query and turn it into an iterable of pairs *measurement_name, dictionary of a line of the measurement*.
- *parse_statistics*: extends the *parse_influx* function and turn its iterable into `Scenario` instances; requires that the OpenBACH tags are included in the InfluxDB response.
- *parse_orphans*: extends the *parse_influx* function and turn its iterable into a `Scenario` instance; do not try to extract tags out of each row of data and consider every column as a statistic.
- *line_protocol*: generate chunks of body from a `Job` instance, ready to be imported into InfluxDB through the *data_write* method.

It also provide the `Condition` classes hierarchy that can be used as the *condition* parameter in the *timestamps*, *scenarios*, *remove_statistics*, and *orphans* methods:

- `Operator` enum that hold comparison operators usable in InfluxDB
- `ConditionField(name, operator, value)` to compare a field value to the given one (example: `ConditionField("throughput", Operator.GreaterThan, 100)` or `ConditionField("status", Operator.Matches, "Failed")`)
- `ConditionTag(name, operator, value)` to compare a tag value to the given one (example: `ConditionTag("@scenario_instance_id", Operator.Equal, 1234)`); automatically used by the aforementioned methods to wrap the optional arguments.
- `ConditionTimestamp(operator, value, unit='ms', frow_now=False)` to compare the time of a record to the given value (example: `ConditionTimestamp(Operator.LowerThan, 123456789)`); automatically used by the aforementionned methods to wrap the optional *timestamps* arguments through the `ConditionTimestamp.from_timestamps` classmethod.
- `ConditionAnd(*conditions)` construct a condition that must match all provided conditions to succeed.
- `ConditionOr(*conditions)` construct a condition that must match any provided condition to succeed.

ElasticSearch Utilities

On top of the common methods, `ElasticSearchConnection` provides a few more specific methods:

- *search_query*: base method to send a POST request to ElasticSearch in order to retrieve data; implements scrolled querying so overly long queries are split into smaller chunked results. Generate results into an iterable. Accepts the query dictionary as parameter as well as optional query-string parameters to shorten the amount of data returned from matched documents and returns the JSON data that ElasticSearch sent back.
- *delete_query*: base method to send a POST request to ElasticSearch in order to delete data; accepts the query dictionary as parameter and returns the JSON data that ElasticSearch sent back.
- *data_write*: base method to send a POST request to ElasticSearch in order to create/update data; accepts the raw request body string as parameter and returns the response.

The query dictionaries must conform to the ElasticSearch Query DSL

The `data_access.elasticsearch_tools` also provide some utility functions:

- *tags_to_query*: create an ElasticSearch query from some OpenBACH tags.
- *parse_logs*: accepts the raw JSON from an ElasticSearch search query and turn it into Scenario instances; requires that the OpenBACH tags are included in the ElasticSearch response.
- *parse_orphans*: accepts the raw JSON from an ElasticSearch search query an populate a Log instance from it; only uses records that does **not** include any OpenBACH tag.
- *rest_protocol*: generate chunks of body from a Log instance and some metadata, ready to be imported into ElasticSearch through the *data_write* method.

Result Scenarios

Scenario objects

`Scenario` objects hold the state of a single scenario instance. It can also hold the state of sub-scenarios if they exist.

The following attributes are used to read data from a `Scenario` instance:

- *instance_id*: the numerical ID of this scenario instance
- *owner_instance_id*: the numerical ID of the top-level scenario that ultimately lead to running this instance (most often equal to `instance_id`)
- *own_jobs*: iterable of `Job` instances launched directly by this scenario
- *jobs*: iterable of `Job` instances recursively launched by this scenario and all its sub-scenarios
- *own_scenarios*: iterable of `Scenario` instances corresponding to sub-scenarios launched directly by this scenario
- *scenarios*: iterable of `Scenario` instances corresponding to sub-scenarios recursively launched by this scenario and its sub-scenarios
- *own_agents*: iterable of `Agent` instances that groups jobs launched by this scenario on each agent they were launched
- *agents*: iterable of `Agent` instances that recursively groups jobs launched by this scenario and all its sub-scenarios on each agent they were launched

Job objects

`Job` objects holds the state of a single job instance. They group statistics emitted by the job as well as logs it generated.

The following attributes are used to read the state of a `Job`:

- *name*: The name of the job
- *instance_id*: The numerical ID of the job
- *agent*: The name of the agent where this job instance was run
- *logs_data*: A `Log` instance holding logs sent by this job
- *statistics*: A `Statistic` instance holding data generated by this job without suffix. Can also be called as a method to specify the desired suffix (*e.g.* `job.statistics(suffix='Flow1')`).
- *suffixes*: an iterable of suffixes names that were generated by this job.

Statistics objects

A collection of data generated by a single job instance under a given suffix.

`Statistic` objects store their data into the `dated_data` instance attribute which is a dictionary:

- keys are timestamp
- values are a dictionary of statistic name and their associated value for the relevant timestamp

Log objects

A collection of logs generated by a single job instance.

Log objects store their data into the numbered_data instance attribute which is a dictionary:

- keys are IDs of the logs in the ElasticSearch database
- values are a `_LogEntry` instances which provide the following attributes:
  - *_id*
  - *_index*
  - *_type*
  - *_timestamp*
  - *_version*
  - *facility*
  - *facility_label*
  - *host*
  - *logsource*
  - *message*
  - *pid*
  - *priority*
  - *severity*
  - *severity_label*

Utility functions

Each of the described classes have a `.json` attribute that can serialize the content of an instance into a dictionary suitable to be turned into a JSON string by the `json` module. They also have a `.load(data)` class-method that can re-create an instance directly from such dictionary. The `read_scenario(filename)` utility function is a shortcut to recreate a `Scenario` instance from a file containing such JSON data.

The described classes also define some `get_or_create_*` functions that rely on the `_get_or_create(container, constructor, *key, args=None)` utility function: if the container does not contain the requested key, it will create it by calling the constructor with the unpacked `args` (if `args` is `None`, it uses `key` instead); it then returns `container[key]`. This utility function is used to implement the following method that help populate various instances:

- `Scenario.get_or_create_subscenario(self, instance_id)`
- `Scenario.get_or_create_job(self, name, instance_id, agent)`
- `Agent.get_or_create_job(self, name, instance_id)`
- `Job.get_or_create_statistics(self, suffix=None)`

The `get_or_create_scenario(scenario_id, cache)` utility function also make use of this mechanism to help maintain a dictionary (`cache`) of `Scenario` instances and ease building the owner/subscenario hierarchy.

Lastly, `extract_jobs(scenario)` is a utility generator that is alike to `scenario.jobs` but instead of `Job` instances yields triplets of `(scenario_instance_id, owner_scenario_instance_id, Job_instance)`.

Example

The following example demonstrate simple usage of data retrieval and import:

transfer_data.py

```
from data_access import CollectorConnection


def main(src_address, dest_address, scenario_id):
    src_collector = CollectorConnection(src_address)
    scenario, = src_collector.scenarios(scenario_instance_id=scenario_id)  # Note the
comma as we're unpacking a generator

    dest_collector = CollectorConnection(dest_address)
    dest_collector.import_scenario(scenario)


 if __name__ == '__main__':
    src = input('Address of the collector to extract data from: ')
    dest = input('Address of the collector to import data into: ')
    scenario = input('Scenario instance ID to transfer between collectors: ')
    main(src, dest, scenario)
```