routingfilter

Release 1.0

Jul 25, 2023

Contents:

1 Usage 1

2 Available filters 3

3 Example 5

4 Routing 7

5 Indices and tables 9

6 Keywords 11

7 Benchmark tests 13

Python Module Index 15

Index 17

CHAPTER 1

Usage

Sample usage:

```
from routingfilter.routing import Routing

test_event_1 = {
    "tags": "mountain_bike",
    "wheel_model": "Superlight",
    "frame": "aluminium",
    "gears": "1x12",
    "suspension": "full"
}

test_rule_1 = {
    "streams": {
        "rules": {
            "mountain_bike": [
                {
                    "filters": [
                        {
                            "type": "EQUALS",
                            "key": "wheel_model",
                            "description": "Carbon fiber wheels needs manual truing",
                            "value": ["Superlight", "RacePro"]
                        }
                    ],
                    "streams": {
                        "Workshop": {
                            "workers_needed": 1
                        }
                    }
                }
            ]
        }
    }
```

(continues on next page)

(continued from previous page)

```
}

routing = Routing()
routing.load_from_dict([test_rule_1])
routing.match(test_event_1)
```

The rule's top level name (default *streams*) can be changed. In this case, the new name can be specified when calling the *match* method.

At the moment, the second level (*rule*) is hard-coded and will likely be removed in the future, since it has no semantic.

The third level matches a field in the event (default *tags*). It can be changed when calling the *matches* method. When the first rule with a given tag (i.e. "mountain_bike") matches the event, the following are ignored. Rules with different tags can match independently (for example, if we want to send an event to different pipelines, based on the tag).

The "streams" element after *filters* means that, if the filter matches, the event will be enriched with the *Workshop* dictionary.

CHAPTER 2

Available filters

For filter types which use “key” and “value” field, they can be both a string or a list of strings. The chosen logic is OR (at least a match must be satisfied).

- **ALL** - matches with everithing, always returns True (if this matches, all other rules are ignored)
- **EXISTS** - returns True if the key in “key” field exists
- **NOT_EXISTS** - returns False if the key in “key” field exists
- **EQUALS** - returns True if the value in the specified “key” is equal to “value”
- **STARTSWITH** - returns True if a “key“‘s value starts with “value”
- **ENDSWITH** - returns True if a “key“‘s value ends with “value”
- **KEYWORD** - returns True if “value” is present in “key” (item in list or string in substring)
- **REGEXP** - returns True if a “key“‘s value matches the RegExp specified in “value”
- **NETWORK** - Parses the field into ad IP address or network and returns True if the IP address is contained in the specified network
- **NOT_NETWORK** - Parses the field into ad IP address or network and returns True if the IP address is NOT contained in the specified network
- **DOMAIN** - Similar to EQUALS but also tries to parse subdomains (separated by “.”)
- **GREATER** - returns True if the value in the specified “key” is greater than “value”.
- **LESS** - returns True if the value in the specified “key” is less than “value”
- **GREATER_EQ** - returns True if the value in the specified “key” is greater than or equal to “value”
- **LESS_EQ** - returns True if the value in the specified “key” is less than or equal to “value”
- **TYPE_OF** - returns True if the target type is the same of the value. Possible types checked are: str, int, bool, list, dict, ip address or mac address

The filters NETWORK and NOT_NETWORK must be strings containing a valid IP or network address (using CIDR notation), otherwise a ValueError is raised. The filters GREATER, LESS, GREATER_EQ, LESS_EQ require float (or float-parsable) values, otherwise a ValueError is raised.

CHAPTER 3

Example

Let's see a routing application with firewall rules. We have network traffic events in the following format:

```
test_event_n1 = {
  "tags": "ip_traffic",
  "src_addr": "192.168.1.10",
  "dst_addr": "192.168.1.15",
  "domain": "test.domain.local"
}
test_event_n2 = {
  "tags": "ip_traffic",
  "src_addr": "192.168.2.10",
  "dst_addr": "192.168.2.15",
  "domain": "test.otherdomain.local"
}
```

We want to filter all traffic tagged as *ip_traffic*, coming from the subnet *192.168.1.0/24* and enrich all the other events with a new field *processed*. We create the following rule:

```
test_rule_n1 = {
  "streams": {
    "rules": {
      "ip_traffic": [
        {
          "filters": [
            {
              "type": "NETWORK",
              "key": "src_addr",
              "value": "192.168.1.0/24"
            }
          ],
          "streams": {
            "filtered": False
          }
        }
```

(continues on next page)

(continued from previous page)

```
      ]
    }
  }
}
```

If we apply this rule to *test_event_n1*, it will match, since the *src_addr* is in the subnet *192.168.1.0/24*. We are obtaining the following output:

```
{
  "rules": [
    {
      "type": "NETWORK",
      "key": "src_addr",
      "value": "192.168.1.0/24"
    }
  ],
  "output": {
    "filtered": False
  }
}
```

The second event will not match with rule *test_event_n1*, since the *src_addr* is not in the subnet *192.168.1.0/24*. The function will return an empty dictionary.

CHAPTER 4

Routing

class routingfilter.routing.**Routing**

Bases: object

get_rules() → Optional[dict]

Return the currently loaded rules. It is mainly used for debugging purposes.

Returns A dict or None

Return type Optional[dict]

load_from_dicts(*rules_list: List[dict], validate_rules: bool = True, variables: Optional[dict] = None*) → None

Load routing configuration from a dictionary. It merges the different rules in list into a single routing rule. It optionally performs some rules validation before accepting them (an exception is raised in case of errors).

Parameters

- **rules_list** (*List[dict]*) – The configuration
- **validate_rules** (*bool*) – Perform rules validation (default=True). It can be disabled to improve performance (unsafe)
- **variables** (*Optional[dict]*) – Variables dictionary to replace rule values

Return type None

load_from_jsons(*rules_list: List[str], validate_rules: bool = True, variables: Optional[dict] = None*) → None

Load routing configuration from JSON data. It merges the different rules in list into a single routing rule. It optionally performs some rules validation before accepting them (an exception is raised in case of errors).

Parameters

- **rules_list** (*List[str]*) – The json data, which will be parsed into a dict
- **validate_rules** (*bool*) – Perform rules validation (default=True). It can be disabled to improve performance (unsafe)
- **variables** (*Optional[dict]*) – Variables dictionary to replace rule values

Return type None

match (*event: dict*, *type_: str = 'streams'*, *tag_field_name: str = 'tags'*) → List[dict]

Process a single event message through routing filters and verify if it matches with (at least) one filter. For each top level tag in the rule, only the first matching filter is returned. Multiple dictionaries can only be returned with rules matching different tags.

Parameters

- **event** (*dict*) – The entire event to process
- **type** (*str*) – The event type (can be 'streams', 'customer' or everything else, as defined in the routing config). If the type does not exists, an empty list is returned
- **tag_field_name** (*str*) – The event field to search into (default='tags')

Returns A list of dicts containing the matched rules and the outputs in the following format: {"rules": [...], "output": {...}}; an empty list if no rule matched

Return type List[dict]

rule_in_routing_history (*type_*, *event*, *rule*)

Checking if the given rule has already been processed

Parameters

- **type** (*dict*) – The **type_** of the event
- **event** (*dict*) – The entire event to process
- **rule** (*dict*) – The rule to check

CHAPTER 5

Indices and tables

- genindex
- modindex
- search

CHAPTER 6

Keywords

It is possible to use keywords in the routing, in order to use variables into the filter values. Keywords are strings that start with *$*, for example: $BICYCLE_COLORS. The definition of keywords have to be included in a *.json* file into a *dictionaries* directory, for example:

```
{
  "BICYCLE_COLORS": ["red","blue"]
}
```

CHAPTER 7

Benchmark tests

The benchmark tests were developed in order to analyze the routing execution time. The tests are available for the EQUALS, STARTSWITH, ENDSWITH, KEYWORD, REGEXP, NETWORK, DOMAIN and GREATER filters. In particular, the tests are: * test1_<FILTER>_no_key_match: it sends 100 messages with 50 fields, but without the *wheel_model* key * test2_<FILTER>_key_exists: it sends 100 messages with 50 fields and one of the keys is *wheel_model*, but with a value different from *Superlight* * test3_<FILTER>_list_values: it sends 100 messages with 50 fields and one of the keys is *wheel_model*. In addition, the rule contains 100 values * test4_<FILTER>_values_message: both the *wheel_model* key in the event and the rule include a list of 100 values

In order to launch the benchmark tests, run

```
python routing_benchmark.py
```

Python Module Index

r

routingfilter.routing, 7

Index

G

`get_rules()` (*routingfilter.routing.Routing method*), 7

L

`load_from_dicts()` (*routingfilter.routing.Routing method*), 7
`load_from_jsons()` (*routingfilter.routing.Routing method*), 7

M

`match()` (*routingfilter.routing.Routing method*), 8

R

`Routing` (*class in routingfilter.routing*), 7
`routingfilter.routing` (*module*), 7
`rule_in_routing_history()` (*routingfilter.routing.Routing method*), 8