REST API

Version v1.0, 04.07.2020: First book

Table of contents

Introduction

The REST API for the data historian is currently exposing the following enpoints:

- */api/historian/v0* : The main API of the historian which currently only contains endpoints for importing and exporting data.
- POST */api/historian/v0/import/csv*: Un endpoint pour importer des points depuis des fichiers csv.
- POST */api/historian/v0/import/json*: An endpoint to import points sent in the body of the request, in json format .
- POST */api/historian/v0/export/csv*: An endpoint to export points in csv format.
- GET */api/grafana/v0*: The API to interact with Grafana using the hurence "Hurence-Historian" datasource plugin.
- POST */api/grafana/v0/query*: Get the points of the historian.
- POST */api/grafana/v0/search*: Get the name of the metrics stored in the historian.
- POST */api/grafana/v0/search/tags*: Get the different tag names in historian.
- POST */api/grafana/v0/search/values*: Get the values for specified field (the field can be a tag name).

Detailed descriptions of the API endpoints

Main api

Data injection

The following endpoints are used to inject data points in the historian :

POST /api/historian/v0/import/csv

General description

This request allows to inject data points from CSV files. The format of the http request must be Content-Type: multipart/form-data;

Table 1. Request parameters

attribute	multivalued	description	mandatory	possible values	default value
my_csv_file	Yes	the path of the csv files to import. Please note that each file is imported individually independently. In other words, listing several files or making a request per file is equivalent. The settings of the request are common to all the files listed in the request. The files must contain a header.	Yes		
mapping.name	No	the name of the column that corresponds to "name". It is the main identifier of a metric.	No	must be a column name from the header of the attached csv files.	"metric"
mapping.value	No	the name of the column that corresponds to "value". It is the point value. The values in this column must be numeric values.	No	must be a column name from the header of the attached csv files.	"value"
mapping.timestamp	No	The name of the header column which corresponds to "timestamp". It is the date of the point. By default this column must contain an epoch timestamp in milliseconds (see format_date)	No	must be a column name from the header of the attached csv files.	"timestamp"

attribute	multivalued	description	mandatory	possible values	default value
mapping.quality	No	The name of the header column which corresponds to "quality". It is the quality of the point (quality is currently not stored in the historian - next release will provide quality)	No	must be a column name from the header of the attached csv files.	"quality"
mapping.tags	Yes	The names of the header columns to be considered as a tag. Tags are concepts that describe the metric in addition to its name ("name" column). All columns not entered in the mapping will be ignored during the injection.	No	must be a column name from the header of the attached csv files.	
format_date	No	The expected format for the values in the "timestamp" column.	No	Must be a value in: [MILLISECONDS_EPOCH,SECONDS_EPOCH,MICROSECONDS_EPOCH,NANOSECONDS_EPOCH] NANOSECONDS_EPOCH : the value must be the number of nanoseconds since January 1, 1970 UTC. MICROSECONDS_EPOCH : the value must be the number of microseconds since January 1, 1970 UTC. MILLISECONDS_EPOCH : the value must be the number of milliseconds since January 1, 1970 UTC. SECONDS_EPOCH : the value must be the number of seconds since January 1, 1970 UTC. Or another value, in this case this value must be a valid date format, for example "yyyy-mm-dd".	"MILLISECONDS_EPOCH"
timezone_date	No	the timezone for the dates in the csv.	No	must be a valid timezone.	"UTC"

attribute	multivalued	description	mandatory	possible values	default value
group_by	Yes	This parameter is very important! If it is misused it is possible to corrupt already existing data. List all the fields that will be used to build the chunks here. By default we group the points in chunk only according to their name. However, it is also possible to group according to the value of some tags. Be aware that this will impact the way the data will be retrieved later. For example if we inject data by grouping by "name" and the tag "factory". Then we can request the values for each factory by filtering on the correct value of the factory tag (see the API to request the points). Only if thereafter someone injects other data without grouping by the tag "factory" then the data will find themselves mixed.	No	Accepted values are "name" (whatever the mapping, use "name" and not the name of the column in the csv). Otherwise the other accepted values are the column names added as a tag.	"name"

- If there is a problem with the request we receive a 400 BAD_REQUEST response.
- If everything went well we receive a 201 CREATED response:

Table 2. Response description

json path	type	description
tags	array	the tags entered in the request.
grouped_by	array	the fields that are used for the group by (completed in the request).
report	json object	a report on the chunks that were injected.

Examples

======= Example 1

Example of a minimal curl command :

```
curl -X POST \
http://localhost:8080/api/historian/v0/import/csv \
-F 'my_csv_file=@/path/de/mon/csv/file.csv'
```

Content of the csv file used :

```
metric,timestamp,value
metric_1, 1, 1.2
metric_1, 2, 2
metric_1, 3, 3
```

Here the answer :

```
{
    "tags" : [ ],
    "grouped_by" : [ "name" ],
    "report" : [
        {
            "name" : "metric_1",
            "number_of_points_injected" : 3,
            "number_of_point_failed" : 0,
            "number_of_chunk_created" : 1
        }
    ]
}
```

======= Example 2

Example of a curl command :

```
curl -X POST \
http://localhost:8080/api/historian/v0/import/csv \
-F 'my_csv_file=@/path/de/mon/csv/file.csv' \
-F mapping.name=metric_name_2 \
-F mapping.value=value_2 \
-F mapping.timestamp=timestamp \
-F mapping.quality=quality \
-F mapping.tags=sensor \
-F mapping.tags=code_install \
-F group_by=name \
-F group_by=tags.sensor \
-F format_date=yyyy-dd-MM HH:mm:ss.SSS \
-F timezone_date=UTC
```

Content of the csv file used :

```
metric_name_2,timestamp,value_2,quality,sensor,code_install
metric_1, 1970-01-01 00:00:00.001, 1.2 ,1.4,sensor_1,code_1
metric_1, 1970-01-01 00:00:00.002, 2 ,1.4,sensor_1,code_1
metric_1, 1970-01-01 00:00:00.003, 3 ,1.4,sensor_2,code_1
metric_2, 1970-01-01 00:00:00.004, 4 ,1.5,sensor_2,code_1
```

Here the answer :

```
{
    "tags" : [ "sensor", "code_install" ],
    "grouped_by" : [ "name", "sensor" ],
    "report" : [ {
        "name" : "metric_1",
        "sensor" : "sensor_1",
        "number_of_points_injected" : 2,
        "number_of_point_failed" : 0,
        "number_of_chunk_created" : 1
    }, {
        "name" : "metric_1",
        "sensor" : "sensor_2",
        "number_of_points_injected" : 1,
        "number_of_point_failed" : 0,
        "number_of_chunk_created" : 1
    }, {
        "name" : "metric_2",
        "sensor" : "sensor_2",
        "number_of_points_injected" : 1,
        "number_of_point_failed" : 0,
        "number_of_chunk_created" : 1
    }]
}
```

======= Example 3

Example of a curl command :

```
curl -X POST \
http://localhost:8080/api/historian/v0/import/csv \
-F 'my_csv_file=@/path/de/mon/csv/file.csv' \
-F 'my_csv_file2=@/path/de/mon/csv/file.csv' \
-F mapping.name=metric_name_2 \
-F mapping.value=value_2 \
-F mapping.timestamp=timestamp \
-F mapping.quality=quality \
-F mapping.tags=sensor \
-F mapping.tags=code_install \
-F group_by=name \
-F group_by=tags.sensor \
-F format_date=yyyy-dd-MM HH:mm:ss.SSS \
-F timezone_date=UTC
```

Content of the csv file used :

```
metric_name_2,timestamp,value_2,quality,sensor,code_install
metric_1, 1970-01-01 00:00:00.001, 1.2 ,1.4,sensor_1,code_1
metric_1, 1970-01-01 00:00:00.002, 2 ,1.4,sensor_1,code_1
metric_1, 1970-01-01 00:00:00.003, 3 ,1.4,sensor_2,code_1
metric_2, 1970-01-01 00:00:00.004, 4 ,1.5,sensor_2,code_1
```

Here the answer :

```
{
  "tags" : [ "sensor", "code_install" ],
  "grouped_by" : [ "name", "sensor" ],
  "report" : [ {
    "name" : "metric_1",
    "sensor" : "sensor_1",
    "number_of_points_injected" : 4,
    "number_of_point_failed" : 0,
    "number_of_chunk_created" : 2
  }, {
    "name" : "metric_1",
    "sensor" : "sensor_2",
    "number_of_points_injected" : 2,
    "number_of_point_failed" : 0,
    "number_of_chunk_created" : 2
  }, {
    "name" : "metric_2",
    "sensor" : "sensor_2",
    "number_of_points_injected" : 2,
    "number_of_point_failed" : 0,
    "number_of_chunk_created" : 2
  } ]
}
```

Note

We notice that we uses twice the same file with the same parameters. If a file is injected several time with the same setup then the chunks will be created only one time. There will have no duplicate in this case.

Reminder

A chunk is created for every metric name, tags that has been chosen and for every day !

======= Example with an empty file

```
curl -X POST \
http://localhost:8080{import-csv-endpoint} \
-F 'my_csv_file=@/path/de/mon/csv/empty.csv' \
```

```
{
    "errors" : [
        {
            "file" : "empty.csv",
            "cause" : "The csv contains Empty header line: can not bind data\n at
[Source: UNKNOWN; line: 1, column: -1] lines which is more than the max number of line
of 5000"
        }
    ]
}
```

======= Example with a file containig just the header

```
curl -X POST \
http://localhost:8080{import-csv-endpoint} \
-F 'my_csv_file=@/path/de/mon/csv/header.csv' \
```

```
{
    "errors" : [
        {
            "file" : "header.csv",
            "cause" : "Empty request body"
        }
    ]
}
```

Troubleshoot

If the file specified in the curl command does not exist, you should have this kind of answers :

```
Warning: setting file path/de/mon/csv/file.csv  failed!
curl: (26) read function returned funny value
```

POST /api/historian/v0/import/json

General description

Allows you to inject points into the historian from a request body in json format. The format of the expected request is a request with Content-Type: application / json;

The body of the expected request is a list of objects. Each object must be composed of the following attributes:

Table 3. Properties of the JSON objects

json path	type	description	mandatory
/name	String	The name of the metric for which we are going to inject the points.	Yes
/points	Array d'array	Points for creating a chunk with the metric name assigned to 'name'. The points are to be completed in the form [[timestsamp<long>, value<double>],..,[timestsamp<long>, value<double>]].	Yes

Examples

Example of a request:

```
[
    {
        "name": "temp",
        "points": [
            [100, 1.0],
            [200, 1.2]
        ]
    },
   {
        "name": "temp_2",
        "points": [
           [100, 1.7],
           [200, 1.9]
        ]
   }
]
```

Example of a minimal request:

```
[
    {
        "name": "temp",
        "points": [
            [100, 1.0]
        ]
    }
]
```

Example of a command curl and its response :

Request :

```
curl -X POST \
  http://localhost:8080/api/historian/v0/import/json \
  -H 'Content-Type: application/json' \
  -d '[
    {
        "name": "temp",
        "points": [
            [100, 1.0],
            [200, 1.2]
        ]
    },
   {
        "name": "temp_2",
        "points": [
           [100, 1.7],
           [200, 1.9]
        ]
   }
]'
```

```
{
    "status": "OK",
    "message": "Injected 4 points of 2 metrics in 2 chunks"
}
```

Errors

======= No valid points

status : BAD REQUEST

```
curl -v -X POST \
  http://localhost:8080/api/historian/v0/import/json \
  -H 'Content-Type: application/json' \
  -d '[
    {
        "name": "temp",
        "points": []
    }
]'
```

```
{
  "error" : "There is no valid points"
}
```

======= empty request body

status : BAD REQUEST

```
curl -v -X POST \
  http://localhost:8080/api/historian/v0/import/json \
  -H 'Content-Type: application/json' \
  -d '[]'
```

```
{
    "error" : "Empty request body"
}
```

======= field point is required

status : BAD REQUEST

```
curl -v -X POST \
  http://localhost:8080/api/historian/v0/import/json \
  -H 'Content-Type: application/json' \
  -d '[
    {
        "name": "temp"
    }
]'
```

```
{
    "error" : "field 'points' is required"
}
```

Data Export

POST /api/historian/v0/export/csv

General description

Allows you to search for points and export them to csv for the desired metrics.

This endpoint uses the same request as the */api/grafana/v0/query* but it gives the response in csv.

Examples

======= Example avec tous les paramètres

Request :

```
curl -X POST \
  http://localhost:8080/api/historian/v0/export/csv \
  -H 'Content-Type: application/json' \
  -d '{
  "from": "2016-10-31T06:33:44.866Z",
  "to": "2020-10-31T12:33:44.866Z",
  "names": ["temp"],
  "max_data_points": 30,
  "sampling":{
     "algorithm": "MIN",
     "bucket_size" : 100
  },
  "tags":{
     "Country": "France"
  },
  "aggregations" : ["MIN", "MAX"],
  "quality":{
     "quality_value": "MIN",
     "quality_agg" : 100
  },
  return_with_quality: true
}'
```

Example of a response :

```
metric,value,date
temp_a,622.1,1477895624866
temp_a,-3.0,1477916224866
temp_a,365.0,1477917224866
temp_b,861.0,1477895624866
temp_b,767.0,1477917224866
```

Interactions with Grafana

The APIs below are used to interact with Grafana using certain data sources (a concept of Grafana). Depending on which plugin version you use(SimpleJson or the more recent Historian one), you will have to use the corresponding API.

Compatible API with the "Hurence-Historian" plugin

GET /api/grafana/v0

This endpoint returns a 200 'OK' response if the historian is working properly.

POST /api/grafana/v0/query

General description

Used to search for points for the desired metrics. Various parameters are available.

Table 4. Parameters for the request

json path	type	description	mandatory	possible values	default value
/names	array	Allows you to enter the metrics for which you want to obtain points.	Yes		
from	String	The start date to search for points. Only points with a date greater than or equal to this date.	No	Must represent a date in the following format (UTC): * yyyy-MM-dd'T'HH: mm: ss.SSS *	January 1, 1970 (UTC)
to	String	The end date to search for points. Only points with a date less than or equal to this date.	No	Must represent a date in the following format (UTC): * yyyy-MM-dd'T'HH: mm: ss.SSS *	The default is infinity
/max_data_points	long	The maximum number of points desired for each metric (Indeed the goal is to draw points on a graph). If necessary the points will be sampled with a default sampling algorithm.	No	positive	1000
/sampling	String	An object which will contain the information of the parameters to be used for the sampling of the points if the number of points to return is greater than the maximum number of points requested.	No		

json path	type	description	mandatory	possible values	default value
/sampling/algorithm	String	L'algorithme à utiliser pour l'échantillonnage	No	les valeurs possibles sont ["NONE", "AVERAGE", "FIRST", "MIN", "MAX"]	AVERAGE (la moyenne des points pour chaque bucket, le timestamp utilisé est celui du premier point du bucket)
/sampling/bucket_size	Int	The size of the buckets desired to sample the data. Please note this size can be changed by the server if this is incompatible with the maximum number of points returned desired.	No		Automatically computed by the server in order to obtain at most the maximum number of points.
/tags	String	Allows you to obtain only the points of the chunks which contain the filled tags. The tags must be indicated in the form of a key / value map.	No		no tags.

Example

======= Minimal Example

Request :

```
curl -v -X POST \
  http://localhost:8080/api/grafana/v0/query \
  -H 'Content-Type: application/json' \
  -d '{
  "names": ["temp"]
}'
```

warn

This request ask for ALL data for the metric "temp".

Response :

```
[
  {
    "name":"temp",
    "datapoints":[
      [1.0,100],
      [1.2,200]
    ],
    "total_points":2
  }
]
```

======= Example with sampling, aggregations, quality in a specific time range.

Request :

```
curl -X POST http://localhost:8080/api/grafana/v0/query -H 'Content-Type:
application/json' \
-d '{
    "from": "1969-10-31T06:33:44.866Z",
    "to": "2030-10-31T12:33:44.866Z",
    "names": ["temp"],
    "max_data_points": 30,
    "sampling":{
        "algorithm": "MIN",
        "bucket_size" : 100
    },
    "aggregations" : ["MIN", "MAX"],
    "return_with_quality": true
}'
```

Response :

```
[
  {
    "name":"temp",
    "datapoints":[
      [1.0,100,1.0]
    ],
    "total_points":1,
    "aggregation":{
      "MIN":1.0,
      "MAX":1.2
    }
  }
]
```

======= Example with tag and quality

Request :

```
curl -X POST \
  http://localhost:8080/api/grafana/v0/query \
  -H 'Content-Type: application/json' \
  -d '{
  "names": ["temp"],
  "max_data_points": 30,
  "tags":{
     "Country": "France"
  },
  "aggregations" : ["MIN", "MAX"],
  "quality":{
     "quality_value": 0.9,
     "quality_agg" : "MIN"
  },
  "return_with_quality": true
}'
```

Response :

TODO does not work at the moment problème avec la section quality ⇒

http://localhost:8983/solr/historian:'rollup(select(search(historian, q=chunk_start:[* TO 9223372036854775807] AND chunk_end:[0 TO **], fl="name,chunk_count,chunk_quality_avg", qt="/export", sort="name asc"), name,chunk_count,chunk_quality_avg,and (eq(name,temp), gteq(chunk_quality_avg,1.0)) as is_quality_ok), over="name,is_quality_ok", sum("chunk_count), count()**)' is not a proper expression clause

POST /api/grafana/v0/search

General description

Lets you search for the different metrics available.

Table 5. Parameter of the request

json path	type	description	mandatory	possible values	default value
/name	String	Part of the name of the metric you are looking for.	No		All metric by default
/limit	Int	The maximum number of metric names to return.	No		100 (configurable)

NOTE	The body of the request is optional.

Example

Request :

```
curl -v -X POST \
  http://localhost:8080/api/grafana/v0/search \
  -H 'Content-Type: application/json' \
  -d '{
  "name": "te",
  "limit": 5
}'
```

Response :

```
["temp","temp_2","temp_no_points"]
```

POST /api/grafana/v0/search/tags

General description

Used to find out the tags name used in historian.

Example

Request :

```
curl -v -X POST \
  http://localhost:8080/api/grafana/v0/search/tags \
  -H 'Content-Type: application/json' \
  -d '{}'
```

or

```
curl -v -X POST \
  http://localhost:8080/api/grafana/v0/search/tags
```

Response :

```
["Country","usIne"]
```

Les valeurs dans "text" sont les valeurs utilisables comme clé dans l'endpoint tag-values.

POST /api/grafana/v0/search/values

General description

Allows you to search for the different values for the indicated key.

Table 6. Parameters of the request

json path	type	description	mandatory	possible values	default value
/field	String	the searched key.	Yes		
/query	String	Part of the name of the key you are looking for.	Yes		No filter
/limit	Int	The maximum number of metric names to return.	No		100

NOTE The query and limit parameters of the query are optional.

Example

Request :

```
curl -v -X POST \
  http://localhost:8080/api/grafana/v0/search/values \
  -H 'Content-Type: application/json' \
  -d '{
  "field": "name",
  "query": "no",
  "limit": 5
}'
```

Response :

```
["temp_no_points"]
```

POST /api/grafana/v0/annotations

Lets you search for annotations.

Example of a request:

```
{
  "from": "2020-2-14T01:43:14.070Z",
  "to": "2020-2-14T06:50:14.070Z",
  "limit" : 100,
  "tags": ["tag1", "tag2"],
  "matchAny": false,
  "type": "tags"
}
```

Example of a response in current release:

```
{
  "annotations" : [
    {
      "time": 1581669794070,
      "text": "annotation 7",
      "tags": ["tag2","tag3"]
    },
    {
      "time": 1581666194070,
      "text": "annotation 6",
      "tags": ["tag3","tag5"]
    }
  ],
  "total_hit" : 2
}
```

Table 7. Parameters for the request

json path	type	description	mandatory	possible values	default value
/from	String	The start date to search for annotations. Only annotations with a date greater than or equal to this date.	No	Must represent a date in the following format (UTC): * yyyy-MM-dd'T'HH: mm: ss.SSS *	January 1, 1970 (UTC)
/to	String	The end date to search for annotations. Only annotations with a date less than or equal to this date.	No	Must represent a date in the following format (UTC): * yyyy-MM-dd'T'HH: mm: ss.SSS *	The default is infinity
/limit	Integer	The maximum number of annotations to return	No	positive integer	100
/tags	Liste de string	The name of the tags to filter the annotations. The behavior depends on the value of the _ / type_ field	No		[] (empty array)
/matchAny	Boolean	If the *type* field is equal to *TAGS*. If *true* the annotations must have at least one of the tags in the *tags* field otherwise the annotations must contain all the tags.	No		true
/type	String	The type of request. If the value is "ALL", all annotations in the time range will be returned. If the type is "TAGS", the annotations must also contain tags (either all or at least one depending on the value of *matchAny*.	No	one value from [ALL, TAGS]	ALL

Compatible API with the "SimpleJson" plugin

GET /api/grafana/simplejson

This endpoint returns a 200 'OK' response if the historian is working properly.

POST /api/grafana/simplejson/query

Used to search for points for the desired metrics. Various parameters are available.

Example of a request:

```
{
    "panelId": 1,
    "range": {
        "from": "2016-10-31T06:33:44.866Z",
        "to": "2016-10-31T12:33:44.866Z",
        "raw": {
            "from": "now-6h",
            "to": "now"
        }
    },
    "rangeRaw": {
        "from": "now-6h",
        "to": "now"
    },
    "interval": "30s",
    "intervalMs": 30000,
    "targets": [
        { "target": "upper_50", "refId": "A", "type": "timeserie" },
        { "target": "upper_75", "refId": "B", "type": "timeserie" }
    ],
    "adhocFilters": [{
        "key": "City",
        "operator": "=",
        "value": "Berlin"
    }],
    "format": "json",
    "maxDataPoints": 550
}
```

Example of a minimal request:

```
{
    "targets": [
      { "target": "upper_50" }
    ]
}
```

Table 8. Parameters for the request

json path	type	description	mandatory	possible values	default value
/targets	array	Allows you to enter the metrics for which you want to obtain points.	Yes		

json path	type	description	mandatory	possible values	default value
/targets/target	String	The name of a metric for which we want to obtain points.	Yes (at least one metric must be listed)		
/range/from	String	The start date to search for points. Only points with a date greater than or equal to this date.	No	Must represent a date in the following format (UTC) : **yyyy-MM-dd'T'HH:mm:ss.SSS**	January 1, 1970 (UTC)
/range/to	String	The end date to search for points. Only points with a date less than or equal to this date.	No	Must represent a date in the following format (UTC): **yyyy-MM-dd'T'HH:mm:ss.SSS**	The default is infinity
/maxDataPoints	long	The maximum number of points desired for each metric (Indeed the goal is to draw points on a graph). If necessary the points will be sampled with a default sampling algorithm.	No	positive	1000
/adhocFilters	String	Used by Grafana	No		

NOTE	The remaining parameters are parameters that are sent by Grafana but which are not currently used.

Example of a response

```
[
  {
    "target":"upper_75",
    "datapoints":[
      [622,1450754160000],
      [365,1450754220000]
    ]
  },
  {
    "target":"upper_90",
    "datapoints":[
      [861,1450754160000],
      [767,1450754220000]
    ]
  }
]
```

POST /api/grafana/simplejson/search

Lets you search for the different metrics available.

Example of a response :

```
{ "target": "upper_50" }
```

Table 9. Parameters for the resquest

json path	type	description	mandatory	possible values	default value
/target	String	Part of the name of the metric you are looking for.	No		

NOTE The body of the request is optional.

Example of a response :

```
["upper_25","upper_50","upper_75","upper_90","upper_95"]
```

POST /api/grafana/simplejson/tag-keys

Used to find out the keys that can be used in the *tag_values* endpoint.

Example of a request:

```
{}
```

Example of a response:

```
[
    {"type":"string","text":"City"},
    {"type":"string","text":"Country"}
]
```

The values in "text" are the values usable as a key in the endpoint tag-values.

Table 10. Parameters for the request

json path	**type**	**description**	**mandatory**	**possible values**	**default value**
/range/from	String	The start date to search for annotations. Only annotations with a date greater than or equal to this date.	No	Must represent a date in the following format (UTC): * yyyy-MM-dd'T'HH: mm: ss.SSS *	January 1, 1970 (UTC)
/range/to	String	The end date to search for annotations. Only annotations with a date less than or equal to this date.	No	Must represent a date in the following format (UTC): * yyyy-MM-dd'T'HH: mm: ss.SSS *	The default is infinity
/limit	Integer	The maximum number of annotations to return	No	positive integer	100
/tags	Liste de string	The name of the tags to filter the annotations. The behavior depends on the value of the _ / type_ field	No		[] (empty array)
/matchAny	Boolean	If the *type* field is worth *TAGS*. If *true* the annotations must have at least one of the tags in the *tags* field, otherwise the annotations must contain all the tags.	No		true
/type	String	The type of request. If the value is "ALL", all annotations in the time range will be returned. If the type is "TAGS", the annotations must also contain tags (either all or at least one depending on the value of *matchAny*).	No	one value from [ALL, TAGS]	ALL

POST /api/grafana/simplejson/tag-values

Lets you search for the different metrics available.

Example of a request:

```
{"key": "City"}
```

Example of a response:

```
[
    {"text": "Eins!"},
    {"text": "Zwei"},
    {"text": "Drei!"}
]
```

Here are the available values for the corresponding keys.

POST /api/grafana/simplejson/annotations

Lets you search for annotations.

Example of a request :

```
{
  "range": {
    "from": "2020-2-14T01:43:14.070Z",
    "to": "2020-2-14T06:50:14.070Z"
  },
  "limit" : 100,
  "tags": ["tag1", "tag2"],
  "matchAny": false,
  "type": "tags"
}
```

Example of the current response:

```
[
  {
    "annotation": "annotation",
    "time": "time",
    "title": "title",
    "tags": "tags",
    "text": "text"
  }
]
```

Table 11. Parameters for the request

json path	type	description	mandatory	possible values	default value
/range/from	String	The start date to search for annotations. Only annotations with a date greater than or equal to this date.	No	Must represent a date in the following format (UTC): * yyyy-MM-dd'T'HH: mm: ss.SSS *	January 1, 1970 (UTC)
/range/to	String	The end date to search for annotations. Only annotations with a date less than or equal to this date.	No	Must represent a date in the following format (UTC): * yyyy-MM-dd'T'HH: mm: ss.SSS *	The default is infinity
/limit	Integer	The maximum number of annotations to return	No	positive integer	100
/tags	Liste de string	The name of the tags to filter the annotations. The behavior depends on the value of the _ / type_ field	No		[] (empty array)
/matchAny	Boolean	If the *type* field is worth *TAGS*. If *true* the annotations must have at least one of the tags in the *tags* field otherwise the annotations must contain all the tags.	No		true
/type	String	The type of request. If the value is "ALL", all annotations in the time range will be returned. If the type is "TAGS", the annotations must also contain tags (either all or at least one depending on the value of *matchAny*).	No	one value from [ALL, TAGS]	ALL