REST API

Version v1.0, 04.07.2020: First book

Table des matières

Introduction

L'api REST de l'historian est actuellement composée des endpoints ci-dessous :

- */api/historian/v0* : L'api principale de l'historian qui ne contient actuellement que des endpoints pour l'import et l'export de données.
- POST */api/historian/v0/import/csv*: Un endpoint pour importer des points depuis des fichiers csv.
- POST */api/historian/v0/import/json*: Un endpoint pour importer des points au format json envoyé dans le corps de la requête.
- POST */api/historian/v0/export/csv*: Un endpoint pour exporter des points au format csv.
- GET */api/grafana/v0*: L'api pour interagir avec Grafana en utilisant le plugin datasource de hurence "Hurence-Historian".
- POST */api/grafana/v0/query*: Permet d'obtenir les points de l'historian.
- POST */api/grafana/v0/search*: Permet d'obtenir les noms des métriques de l'historian.
- POST */api/grafana/v0/search/tags*: Permet d'obtenir le nom des tags existants dans l'historian.
- POST */api/grafana/v0/search/values*: Permet d'obtenir les valeurs pour le champ indiqué (le champs peut être un nom de tag).

Description détaillée des endpoints

Api principal

Ingestion de données

Ces endpoints servent à injecter des points dans l'historian :

POST /api/historian/v0/import/csv

Description Générale

Permet d'injecter des points dans l'historian à partir de fichiers csv. Le format de la requête attendue est une requête avec Content-Type: multipart/form-data;

Table 1. Paramètres de la requête

attribut	multi valué	description	requis	valeurs possibles	valeur par défaut
<a_name>	Oui	le chemin des fichier csv à importer. Attention chaque fichier est importé individuellement de manière indépendante. Autrement dit, joindre plusieurs fichiers ou faire une requête par fichier est équivalent. Les paramètres de la requêtes sont communs à tous les fichiers listés dans la requête. Les fichiers doivent obligatoirement contenir un header.	Oui		
mapping.name	Non	le nom de la colonne qui correspond à "name". C'est l'identifiant principal d'une métrique.	Non	doit être un nom de colonne du header des fichiers csv joints.	"metric"
mapping.value	Non	le nom de la colonne qui correspond à "value". C'est la valeur du point. Les valeurs de cette colonne doivent être des valeurs numériques.	Non	Doit être un nom de colonne du header des fichiers csv joints.	"value"
mapping.timestamp	Non	Le nom de la colonne du header qui correspond à "timestamp". C'est la date du point. Par défaut cette colonne doit contenir un timestamp epoch en milliseconde (voir format_date)	Non	Doit être un nom de colonne du header des fichiers csv joints.	"timestamp"

attribut	multi valué	description	requis	valeurs possibles	valeur par défaut
mapping.quality	Non	Le nom de la colonne du header qui correspond à "quality". C'est la qualité du point (pour le moment la qualité n'est pas stockée dans l'historian)	Non	Doit être un nom de colonne du header des fichiers csv joints.	"quality"
mapping.tags	Oui	Les noms des colonnes du header à considérer comme un tag. Les tags sont des notions qui décrivent la métrique en plus de son nom (colonne "name"). Toutes les colonnes non renseignées dans le mapping seront ignorées lors de l'injection. TODO expliquer on prend le first	Non	Doit être un nom de colonne du header des fichiers csv joints.	
format_date	Non	Le format attendu pour les valeurs de la colonne "timestamp".	Non	Doit être soit une valeur parmi: [MILLISECONDS_EPOCH,SECONDS_EPOCH,MICROSECONDS_EPOCH,NANOSECONDS_EPOCH] NANOSECONDS_EPOCH : la valeur doit être le nombre de nanosecondes depuis le 1 janvier 1970 UTC. MICROSECONDS_EPOCH : la valeur doit être le nombre de microsecondes depuis le 1 janvier 1970 UTC. MILLISECONDS_EPOCH : la valeur doit être le nombre de millisecondes depuis le 1 janvier 1970 UTC. SECONDS_EPOCH : la valeur doit être le nombre de secondes depuis le 1 janvier 1970 UTC. Soit une autre valeur, dans ce cas cette valeur doit être un format de date valide, par exemple "yyyy-MM-dd".	"MILLISECONDS_EPOCH"
timezone_date	Non	le timezone pour les dates dans le csv.	Non	doit être un timezone valide.	"UTC"

attribut	multi valué	description	requis	valeurs possibles	valeur par défaut
group_by	Oui	Ce paramètre est très important ! Si il est mal utilisé il est possible de corrompre les données déjà existantes. Il faut lister ici tous les champs qui vont être utilisés pour construire les chunks. Par défaut on groupe les points en chunk uniquement en fonction de leur nom. Cependant il est possible de grouper également en fonction de la valeur de certains tags. Il faut savoir que cela va impacter la manière dont les données seront récupérées par la suite. Par exemple si on injecte des données en groupant par "name" et le tag "usine". Alors on pourra requêter les valeurs pour chaque usine en filtrant sur la bonne valeur du tag usine (voir l'api pour requêter les points). Seulement si par la suite quelqu'un injecte d'autres données sans grouper par le tag "usine" alors les données vont se retrouver mélangées.	Non	Les valeurs acceptées sont "name" (peu importe le mapping utilisez "name" et non le nom de la colonne dans le csv). Sinon les autres valeurs acceptées sont les noms de colonnes ajoutées comme tag.	"name"

- S'il y a un problème avec la requête on reçoit une réponse 400 BAD_REQUEST.
- Si tout s'est bien passé on reçoit une réponse 201 CREATED :

Table 2. Description de la réponse

json path	type	description
tags	array	les tags renseignés dans la requête.
grouped_by	array	les champs qui sont utilisés pour le group by (renseignés dans la requête).
report	json object	un rapport sur les chunks qui ont été injectés.

Exemples

======= Exemple 1

Exemple de commande curl minimal :

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

Contenu du fichier csv utilisé :

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

La réponse est la suivante :

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

======= Exemple 2

Exemple de commande curl :

```
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
```

Contenu du fichier csv utilisé :

```
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
```

La réponse est la suivante :

```
{
    "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
    }]
}
```

======= Exemple 3

avec la requête suivante :

```
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
```

Contenu du fichier csv utilisé :

```
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
```

on reçoit cette réponse :

```
{
  "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

On remarquera qu'on a utiliser le même fichier deux fois avec les même paramètres. Si un même fichier est injecter plusieurs fois avec les même paramètres alors l'import du deuxième fichier écrase les chunks du premier. Il n y a donc pas de doublons.

Rappel

Un chunk est créer par métrique et par la valeurs des tags sélection mais aussi pour chaque changement de jour !

======= Exemple avec un fichier vide

```
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"
        }
    ]
}
```

======= Exemple avec un fichier ayant juste un 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

Si le fichier spécifier dans la commande curl n'existe pas, vous devriez avoir une réponse comme celle-la :

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

POST /api/historian/v0/import/json

Description Générale

Permet d'injecter des points dans l'historian à partir d'un corps de requête au format json. Le format de la requête attendue est une requête avec Content-Type: application/json;

Le corps de la requête attendue est une liste json d'objets. Chaque objet doit être composé des attributs suivants :

Table 3. Propriété des objets json

json path	type	description	requis
/name	String	Le nom de la métrique pour laquelle on va injecter les points.	Oui
/points	Array d'array	Les points pour créer un chunk avec le nom de métrique assigné à 'name'. Les points sont a renseigner sous la forme [[timestsamp<long>, value<double>],..,[timestsamp<long>, value<double>]].	Oui

Exemples

Exemple d'une requête (le body) :

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

Exemple d'une requête minimal (le body) :

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

Exemple d'une requête avec curl et de sa réponse :

```
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"
}
```

Export de données

POST /api/historian/v0/export/csv

Description Générale

Permet de rechercher des points et les exporter en csv pour les métriques désirées.

Ce endpoint utilise la même requête que l'endpoint */api/grafana/v0/query* mais il donne la réponse en csv.

Exemples

======= Exemple avec tous les paramètres

Requête :

```
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
}'
```

Exemple de réponse :

```
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 avec Grafana

Les api ci-dessous sont utilisées pour interagir avec Grafana en utilisant certaines datasources (une notion de Grafana).

Api compatible avec le plugin "Hurence-Historian"

GET /api/grafana/v0

Cet endpoint renvoie une réponse 200 'OK' si l'historian fonctionne correctement.

POST /api/grafana/v0/query

Description Générale

Permet de rechercher des points pour les métriques désirées. Divers paramètres sont disponible.

Table 4. Paramètres de la requête

json path	type	description	requis	valeurs possibles	valeur par défaut
/names	array	Permet de renseigner les métriques pour lesquelles on souhaite obtenir des points.	Oui		
from	String	La date de début pour rechercher les points. Il ne sera retourné que les points avec une date supérieure ou égale à cette date.	Non	Doit réprésenter une date au format suivant (UTC) : **yyyy-MM-dd'T'HH:mm:ss.SSS**	Le 1 Janvier 1960 (UTC)
to	String	La date de fin pour rechercher les points. Il ne sera retourné que les points avec une date inférieure ou égale à cette date.	Non	Doit réprésenter une date au format suivant (UTC) : **yyyy-MM-dd'T'HH:mm:ss.SSS**	La valeur par défaut est l'infini
/max_data_points	long	Le nombre maximum de point désirés pour chaque métrique (En effet le but est de tracer des points sur un graphe). Si nécessaire les points seront échantillonnés avec un algorithme de sampling par défaut.	Non	positif	1000
/sampling	String	Un objet qui va contenir les information des paramètres à utiliser pour l'échantillonage des points si le nombre de points à retourner est supérieur au nombre maximum de point demandé.	Non		

json path	type	description	requis	valeurs possibles	valeur par défaut
/sampling/algorithm	String	L'algorithme à utiliser pour l'échantillonnage	Non	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	La taille des buckets souhaité pour échantillonner les données. Attention cette taille peut être changée par le serveur si cela est incompatible avec le nombre maximum de point retournés souhaité.	Non		Calculé automatiquement par le serveur afin d'obtenir au plus le nombre maximum de points.
/tags	String	Permet d'obtenir uniquement les points des chunks qui contiennent les tags renseignés. Les tags doivent être indiqués sous la forme d'une map de clés / valeurs.	Non		Pas de tags.

Exemple

======= Exemple Minimal

Exemple de requête :

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

Attention cette requête demande TOUT les points pour la métrique "temp".

Exemple de réponse :

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

======= Exemple avec sampling, agrégations, qualité dans une plage de temps

Exemple de requête :

```
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
}'
```

Exemple de réponse :

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

======= Exemple avec tag et qualité

Exemple de requête :

```
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
}'
```

Exemple de réponse :

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

Description Générale

Permet de chercher les différentes métriques disponibles.

Table 5. Paramètres de la requête

json path	type	description	requis	valeurs possibles	valeur par défaut
/name	String	Une partie du nom de la métrique recherché.	Non		Par défault toutes les métriques sont retournés
/limit	Int	Le nombre maximum de noms de métriques à retourner.	Non		100 (paramétrable)

NOTE	Le corps de la requête est optionnel.

Exemple

Requête :

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

Réponse :

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

POST /api/grafana/v0/search/tags

Description Générale

Permet de trouver les noms de tags existants dans l'historian.

Exemple

Requête :

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

ou bien

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

Réponse :

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

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

POST /api/grafana/v0/search/values

Description Générale

Permet de chercher les différentes valeurs pour la clé indiquée.

Table 6. Paramètres de la requête

json path	type	description	requis	valeurs possibles	valeur par défaut
/field	String	la clé recherchée.	Oui		
/query	String	Une partie du nom de la clé recherchée.	Oui		Pas de filtre par défaut
/limit	Int	Le nombre maximum de noms de métriques à retourner.	Non		100

NOTE Le paramètres query et limit de la requête sont optionnels.

Exemple

Requête :

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

Réponse :

```
["temp_no_points"]
```