User Guide

Version v1.0, 02.07.2020: First book

Table of content

Introduction

This guide will help the user of the data historian in understanding the platform, its installation, its use. The data historian having been designed with a view to deployment on a Big Data infrastructure, which by nature are complex environments to implement, we will only discuss in this guide the deployments on a single machine and gradually prepare for the Big One. For deployments on a Big Data multi-node infrastructure with the necessary security, you will have to refer to other documentation or contact Hurence at contact@hurence.com.

Concepts

Hurence Data Historian is a free solution to handle massive loads of timeseries data into a search engine (such as Apache SolR). The key concepts are simple :

- **Measure** is a point in time with a floating point value identified by a name and some tags (categorical features)
- **Chunk** is a set of contiguous Measures with a time interval grouped by a date bucket, the measure name and eventually some tags

The main purpose of this tool is to help creating, storing and retrieving these chunks of timeseries. We use chunking instead of raw storage in order to save some disk space and reduce costs at scale. Also chunking is very usefull to pre-compute some aggregation and to facilitate down-sampling

Data model

A Measure point is identified by the following fields

```
case class Measure(name: String,
                   value: Double,
                   timestamp: Long,
                   year: Int,
                   month: Int,
                   day: String,
                   hour: Int,
                   tags: Map[String,String])
```

A Chunk is identified by the following fields

```
case class Chunk(name: String,
     day:String,
     start: Long,
     end: Long,
     chunk: String,
     count: Long,
     avg: Double,
     stddev: Double,
     min: Double,
     max: Double,
     first: Double,
     last: Double,
     sax: String,
     tags: Map[String,String])
```

As you can see from a Measure points to a Chunk of Measures, the `timestamp` field has been replaced by a `start` and `stop` interval and the `value` is now a base64 encoded string named `chunk`.

In SolR the chunks will be stored according to the following schema (for the current version)

```
<schema name="default-config" version="1.6">
    ...
    <field name="chunk_avg" type="pdouble" multiValued="false" indexed="true" stored=
"true"/>
    <field name="chunk_value" type="string" multiValued="false" indexed="false"/>
    <field name="chunk_count" type="pint" multiValued="false" indexed="true" stored=
"true"/>
    <field name="chunk_day" type="string" multiValued="false" indexed="true" stored=
"true"/>
    <field name="chunk_end" type="plong" multiValued="false" indexed="true" stored=
"true"/>
    <field name="chunk_first" type="pdouble" multiValued="false" indexed="true"
stored="true"/>
    <field name="chunk_hour" type="pint" multiValued="false" indexed="true" stored=
"true"/>
    <field name="chunk_last" type="pdouble" multiValued="false" indexed="true" stored
="true"/>
    <field name="chunk_max" type="pdouble" multiValued="false" indexed="true" stored=
"true"/>
    <field name="chunk_min" type="pdouble" multiValued="false" indexed="true" stored=
"true"/>
    <field name="chunk_month" type="pint" multiValued="false" indexed="true" stored=
"true"/>
    <field name="name" type="string" multiValued="false" indexed="true" stored="true
"/>
    <field name="chunk_outlier" type="boolean" multiValued="false" indexed="true"
stored="true"/>
    <field name="chunk_qualities" type="string" multiValued="true" indexed="true"
stored="true"/>
    <field name="chunk_sax" type="ngramtext" multiValued="false" indexed="true"
stored="true"/>
    <field name="chunk_start" type="plong" multiValued="false" indexed="true" stored=
"true"/>
    <field name="chunk_stddev" type="pdouble" multiValued="false" indexed="true"
stored="true"/>
    <field name="chunk_sum" type="pdouble" multiValued="false" indexed="true" stored=
"true"/>
    <field name="tags" type="text_en" multiValued="true" indexed="true" stored="true
"/>
    <field name="chunk_timestamp" type="plong" multiValued="false" indexed="true"
stored="true"/>
    <field name="chunk_trend" type="boolean" multiValued="false" indexed="true"
stored="true"/>
    <field name="chunk_year" type="pint" multiValued="false" indexed="true" stored=
"true"/>
</schema>
```

Installation

This section describes a simple single node installation of the Hurence Data Historian. The historian is designed for Big Data and therefore it is possible to deploy it on small to very large multi node infrastructures (on multiple racks) with potentially huge storage capacities and in any cases outstanding computing performances.

This type of installation is not described here ; this is the specific domain of experts in designing Big Data infrastructures.

NOTE	Hurence proposes services to install the data historian on Big Data infrastructures for large volumes of data.

In this guide we propose a simple and quick installation on a single node which is ideal for starting. It can be used for testing, developing and storing small volumes data. We would not recommend it for production since in production we will need to prepare the infrastructure to scale and become multi-nodes and have the necessary data redundancy and fail over mechanisms. The second installation is more suitable if you plan to go this route.

- Installation standalone
- Installation de production mono noeud.

Standalone installation

This installation is the one you should use if you are new to the data historian and want to test it. It is not meant for production and large volumes (only a single node) but can evolve if needed towards a true production installation.

Note

Hurence provides assistance to evolve your single node installation towards a proper clusterized production infrastructure.

Pre-requisites for a standalone single node installation

As said earlier this install représents the quickest and easiest way for installing HDH. It is ideal for testing or if you do not have large volumes of data.

The minimal configuration for the server, for this installation is the following:

- OS CentOS or Redhat 7.x
- 16 Gigabits of RAM
- 8 vcores of CPU
- 250 Gigabytes of disk
- Java 8

Installing the Data Historian

Hurence Data Historian is a set of scripts and binary files that let you manupulate time series and chunks of time series.

Download the installation script for your version of data historian at : install.sh.

Run the installation by running this script :

```
bash ./install.sh
```

Some information will be needed. Since we are going to install a single node data historian, you can use the defaults values and just press ENTER for each question.

To access the standalone installation you must type "1". You can skip the Spark installation that is not mandatory. It is useful for more advanced users.

The script is going to download and install packaged and embeded versions of Solr and Grafana.

Make sure you have enough space in the directory where the historian is installed (défault is /opt/hdh) to store the scripts, binaries and the time series data (these data are stored in the same directory by default).

Here is an overview of the installation :

```
Os detected is linux : linux-gnu
Wget is already installed
Where do you want to install Hurence Data Historian ?[/opt/hdh]

Do you want us to install an embedded solr (version 8.2.0 required)? (otherwise you need to have one that can be used from this machine)
1) Yes
2) No
#? 1
Which name to use for the solr collection which will be storing time series ?[historian]

Which name to use for the solr report collection ?[historian-report]

Do you want to add tags names for your time series (you can always add them after installation ?)
1) Add_tags
2) Skip
#? 1


Tag name (STOP when you want stop): tag1
tapped tag tag1
Tag name (STOP when you want stop): stop
array is tag1
Do you want us to install an embedded grafana (version 7.0.3 required)? (otherwise you need to have one that can be used from this machine if you plan to use grafana)
1) Yes
2) No
#? 1
Do you want us to install the historian datasource grafana plugin ? You need it to see data with grafana. We can install it only if grafana is on this machin as single node otherwise you will have to install it manually. If
 you choose to install an embedded grafana you can install it as well.
1) Yes
2) No
#? 1
Do you want us to install an embedded spark (this is not required)?
1) Yes
2) No
#? 2
```

In the following, the variable '$HDH_HOME' will be used to refer to the installation path of the data historian.

After running the script and if you followed the example, you'll get :

- A Solr 8.2 server installed in $HDH_HOME/solr-8.2.0
- The script to start the Solr server. The script has started Solr and you can check that this server is running at : solr UI. You can check the Solr documentation for managing Solr (starting, stopping the Solr server).

- A Grafana server 7.0.3 installed in $HDH_HOME/grafana
- The data source plugin for Grafana datasource Grafana de l'historian is installed to on this server. This plugin is the necessary tooling for visualizing the historian time series in Grafana.
- The Grafana server was started by the script. You can check it is up and running at this address: http://localhost:3000/. You can check the Grafana documentation to interact with Grafana (starting and stopping the server for example).
- The historian server is installed in $HDH_HOME/historian-1.3.5
- A folder "$HDH_HOME/data" has been created to receive the time series data for the historian.

Here is the default structure for $HDH_HOME after the default installation :

- $HDH_HOME/data :folder contening the Solr data (timeseries)
- $HDH_HOME/solr-8.2.0 : folder containing the scripts and binaries for Solr 8.2.0
- $HDH_HOME/solr-8.2.0/bin/solr : script to start and stop Solr
- $HDH_HOME/historian-1.3.5/bin/historian-server.sh : script to start and stop the REST API for the data historian.
- $HDH_HOME/historian-1.3.5/conf/log4j.properties : file to control the level of logs in production mode. (default).
- $HDH_HOME/historian-1.3.5/conf/log4j-debug.properties : file to control the level of logs in debug mode.
- $HDH_HOME/historian-1.3.5/conf/historian-server-conf.json : file that stores the configuration for the API REST server of the data historian.
- $HDH_HOME/application.log : the log file for the data historian.
- $HDH_HOME/grafana-7.0.3 : folder containing the scripts and binaries for Grafana 7.0.3.
- $HDH_HOME/grafana-7.0.3/bin/grafana-server : script for staring and stopping the Grafana server.

When the installation runs successfully, all services are started and the data historian is ready to use.

The following commandes are provided for the case you would need to stop and restart the data historian server.

To start the Hurence Data Historian :

```
$HDH_HOME/historian-1.3.5/bin/historian-server.sh start
```

To stop the Hurence Data Historian :

```
$HDH_HOME/historian-1.3.5/bin/historian-server.sh stop
```

Note

these commands affect neither Grafana nor Solr which are independant services (and need to be started / Stopped separately).

Configuration file for the data historian

The following file is the default configuration. It contains all possible information, some of them are not mandatory :

```
{
  "web.verticles.instance.number": 1,
  "historian.verticles.instance.number": 2,
  "http_server" : {
    "host": "localhost",
    "port" : 8080,
    "historian.address": "historian",
    "debug": false,
    "max_data_points_allowed_for_ExportCsv" : 10000,
    "upload_directory": "/tmp/hurence-historian"
  },
  "historian": {
    "schema_version": "VERSION_0",
    "address" : "historian",
    "limit_number_of_point_before_using_pre_agg" : 50000,
    "limit_number_of_chunks_before_using_solr_partition" : 50000,
    "api": {
      "grafana": {
        "search" : {
          "default_size": 100
        }
      }
    },
    "solr" : {
      "use_zookeeper": true,
      "zookeeper_urls": ["localhost:9983"],
      "zookeeper_chroot" : null,
      "stream_url" : "http://localhost:8983/solr/historian",
      "chunk_collection": "historian",
      "annotation_collection": "annotation",
      "sleep_milli_between_connection_attempt" : 10000,
      "number_of_connection_attempt" : 3,
      "urls" : null,
      "connection_timeout" : 10000,
      "socket_timeout": 60000
    }
  }
}
```

- General conf :

- web.verticles.instance.number : The number of instances of verticles to be deployed to respond to all http requests from clients. A verticle can handle a large number of requests (at least 1000, check the vertx documentation for more information).
- historian.verticles.instance.number : The number of instances of verticles to deploy for the historian service that manages the sampling and the interactions with the backend. This parameter is KEY. In case of performances problem it is likely that augmenting this parameter may help fix the problem.

- Http server conf :
  - http_server/host : the name of the http server to be deployed.
  - http_server/port : the port on which the REST API is to be bound.
  - http_server/historian.address : the name of the deployed vertex historian service. We advice to not change this parameter unless you manage other vertx services. It is important to master vertx when changing this parameter.
  - http_server/max_data_points_allowed_for_ExportCsv : this parameter defines the maximum points the historian will return when a client used the REST export mechanism in CSV format. It is important to set the parameter carefully (not too large and large enough) since all returned data will reside in memory. For large exports we advice using other technics than the REST API call. Parallel processing using Spark is a far better way to export large datasets.
  - http_server/upload_directory : Repertory where csv file uploaded will be stored (temporarily).
- Historian service conf :
  - general conf
    - historian/address : the name of the deployed vertex historian service. We advice to not change this parameter unless you manage other vertx services. This value must be the same as the 'http_server/historian.address'.
    - historian/limit_number_of_point_before_using_pre_agg : this option provides some performance tuning. Take care to not set a too large number.
    - historian/limit_number_of_chunks_before_using_solr_partition : this option provides some performance tuning. Take care to not set a too large number.
    - historian/api/grafana/search/default_size : this option sets the maximum number of metrics to be returned, by default, for the endpoint Search.
    - historian/schema_version : The schema version to use. You should avoid changing this value by hand. It will changed by the system during a rolling update typically.
  - solr conf
    - historian/solr/connection_timeout : the connection timeout in milliseconds to the Solr server.
    - historian/solr/socket_timeout : the connection timeout in milliseconds for all reading sockets on Solr.
    - historian/solr/stream_url : the URL of the Solr collection to use for the stream API of Solr. We recommend to create a dedicated collection (with sufficient resources);

- historian/solr/chunk_collection : the name of the collection where time series are to be stored.
- historian/solr/annotation_collection : the name of the collection where annotations are to be stored.
- historian/solr/sleep_milli_between_connection_attempt : the number of milliseconds to wait between two ping attempts to the Solr server when starting the historian.
- historian/solr/number_of_connection_attempt : the number of attempts in testing connectivity to the Solr server when starting the historian.
- historian/solr/use_zookeeper : in case you are using Solr cloud (with or without a zookeeper server or cluster)
  - option if using zookeeper
    - historian/solr/zookeeper_urls : a list of at list one zookeeper server (ex: ["zookeeper1:2181"]).
    - historian/solr/zookeeper_chroot : the path to the root zookeeper that contains the Solr data. Do not enter or use null if there is no chroot (see the zookeeper documentation).
  - option if zookeeper is not used
    - historian/solr/urls : the http URLs to query Solr. For example ["http://server1:8983/solr", "http://server2:8983/solr"].

Generation of a configuration file, according to entered information, at installation.

Description of the installed historian components

Apache SolR

Apache SolR is the database / search engine used by the historian for storing and indexing time series. It can be replaced by another search engine.

The installation script has installed Solr in the '$HDH_HOME/solr-8.2.0' folder that we will name '$SOLR_HOME' in the following.

It also started to Solr cores locally in the folder `$SOLR_HOME/data`.

To start solr

If you stopped Solr or if after restarting your computer, Solr is not running, you can restart Solr with the following commands :

```
cd $SOLR_HOME
# démarre un core Solr localement ainsi qu'un serveur zookeeper standalone.
bin/solr start -cloud -s $SOLR_HOME/data/solr/node1 -p 8983
# démarre un second core Solr localement qui va utiliser le serveur zookeeper
précédamment créer.
bin/solr start -cloud -s $SOLR_HOME/data/solr/node2/ -p 7574 -z localhost:9983
```

You can check that your Solr instance is up and running by checking on its web UI at this address: (local solr UI

Single node installation for production

Pre-requisite for a single node install in production

The minimal configuration for the server for a single server installation of the data historian (in production) is:

- OS CentOS or Redhat 7.x
- 32 Gigabits of RAM
- 32 vcores of CPU
- 2 Terabytes of disk
- Java 8
- Spark 2.3.4
- Solr 8.2.0
- Grafana 7.0.3

In this section you will find quick guides to help you install Solr 8.2.0, Spark 2.3.4 and Grafana 7.0.3.

We however recommend to refer to the official documentation for all these tools for production installations.

If you have existing servers you can jump to this section

Installing Apache SolR

Apache SolR is the database/search engine used but the data historian. It could be replaced by another search engine.

You can download the 8.2.0 version of Solr from this link: solr-8.2.0.tgz or from this one site officiel.

We invite you to follow the official documentation if you want to install a Solr cluster (and not a single node) in production.

See the commands in next sections for unzip and install of both SolR and Spark

Vérify that your Solr instance is up and running by opening its web user interface at this address: "http://<solrhost>:8983/solr/#/~cloud"

Installing Apache Spark

To install Spark you can download this archive: spark-2.3.4-bin-without-hadoop.tgz

Commands to install Solr and Spark

The following commands allow you to get a local install. For a cluster in production, check the official documentation or get guidance from Hurence.

```
# get Apache Spark 2.3.4 and unpack it
cd $HDH_HOME
wget https://archive.apache.org/dist/spark/spark-2.3.4/spark-2.3.4-bin-without-
hadoop.tgz
tar -xvf spark-2.3.4-bin-without-hadoop.tgz
rm spark-2.3.4-bin-without-hadoop.tgz

# add two additional jars to spark to handle our framework
wget -O spark-solr-3.6.6-shaded.jar
https://search.maven.org/remotecontent?filepath=com/lucidworks/spark/spark-
solr/3.6.6/spark-solr-3.6.6-shaded.jar
mv spark-solr-3.6.6-shaded.jar $HDH_HOME/spark-2.3.4-bin-without-hadoop/jars/
cp $HDH_HOME/historian-1.3.4-SNAPSHOT/lib/loader-1.3.4-SNAPSHOT.jar $HDH_HOME/spark-
2.3.4-bin-without-hadoop/jars/
```

Installing Grafana

You can install Grafana on your platform following this guide: https://grafana.com/docs/grafana/latest/installation/requirements/.

One the Grafana cluster (or single node install) up and running we can install the Grafana plugin for the data historian. This plugin turns our historian to a proper data source and allow the creation and visualisation of dashboards based on the historian.

The minimal requirement for the plugin is the 7.0.3 version of Grafana.

Installing the Grafana data source plugin

To view the historian data using dashboards we make use of Grafana. In this end, we have developed our own Grafana plugins that we update according to new historian or Grafana releases.

To install the data source plugin follow the specific guide installing Grafana datasource plugin

Installing the Hurence Data Historian (HDH)

Hurence Data Historian is a set of scripts and binaries that help you work with time series and chunks. You can download the installation script for your version at : install.sh.

Launch the install with this script:

```
bash ./install.sh
```

You are asked for some configuration information.

Here is an example:

In the following, we will use the '$HDH_HOME' variable to figure out the installation path for the historian.

At the end of the script run, you'll get:

- the HDH installed at the provided path (in $HDH_HOME).
- the Grafana datasource plugin Grafana datasource plugin installed on the Grafana server as indicated at installation

The structure of $HDH_HOME after running the installation script is 'by default':

- $HDH_HOME/bin/historian-server.sh : Script to start and stop the REST API server of the historian
- $HDH_HOME/conf/log4j.properties : File to control the level of logs in production mode (DEFAULT).
- $HDH_HOME/conf/log4j-debug.properties : File to contrôle the level of logs in debug mode.
- $HDH_HOME/conf/historian-server-conf.json : configuration file for the REST API server.

The $HDH_HOME/bin/historian-server.sh script is used to start/stop the REST API server of the historian.

To start the REST API server, type the following command:

```
./bin/historian-server.sh start
```

To stop the REST API server, type the following command:

```
./bin/historian-server.sh stop
```

Note

these commands affect neither Grafana nor Solr that are independant services and need to be started and stopped separately.

Configuration file for the data historian

The following file is the default configuration. It contains all possible information, some of them are not mandatory :

```
{
  "web.verticles.instance.number": 1,
  "historian.verticles.instance.number": 2,
  "http_server" : {
    "host": "localhost",
    "port" : 8080,
    "historian.address": "historian",
    "debug": false,
    "max_data_points_allowed_for_ExportCsv" : 10000,
    "upload_directory": "/tmp/hurence-historian"
  },
  "historian": {
    "schema_version": "VERSION_0",
    "address" : "historian",
    "limit_number_of_point_before_using_pre_agg" : 50000,
    "limit_number_of_chunks_before_using_solr_partition" : 50000,
    "api": {
      "grafana": {
        "search" : {
          "default_size": 100
        }
      }
    },
    "solr" : {
      "use_zookeeper": true,
      "zookeeper_urls": ["localhost:9983"],
      "zookeeper_chroot" : null,
      "stream_url" : "http://localhost:8983/solr/historian",
      "chunk_collection": "historian",
      "annotation_collection": "annotation",
      "sleep_milli_between_connection_attempt" : 10000,
      "number_of_connection_attempt" : 3,
      "urls" : null,
      "connection_timeout" : 10000,
      "socket_timeout": 60000
    }
  }
}
```

- General conf :

- web.verticles.instance.number : The number of instances of verticles to be deployed to respond to all http requests from clients. A verticle can handle a large number of requests (at least 1000, check the vertx documentation for more information).
- historian.verticles.instance.number : The number of instances of verticles to deploy for the historian service that manages the sampling and the interactions with the backend. This parameter is KEY. In case of performances problem it is likely that augmenting this parameter may help fix the problem.

- Http server conf :
  - http_server/host : the name of the http server to be deployed.
  - http_server/port : the port on which the REST API is to be bound.
  - http_server/historian.address : the name of the deployed vertex historian service. We advice to not change this parameter unless you manage other vertx services. It is important to master vertx when changing this parameter.
  - http_server/max_data_points_allowed_for_ExportCsv : this parameter defines the maximum points the historian will return when a client used the REST export mechanism in CSV format. It is important to set the parameter carefully (not too large and large enough) since all returned data will reside in memory. For large exports we advice using other technics than the REST API call. Parallel processing using Spark is a far better way to export large datasets.
  - http_server/upload_directory : Repertory where csv file uploaded will be stored (temporarily).
- Historian service conf :
  - general conf
    - historian/address : the name of the deployed vertex historian service. We advice to not change this parameter unless you manage other vertx services. This value must be the same as the 'http_server/historian.address'.
    - historian/limit_number_of_point_before_using_pre_agg : this option provides some performance tuning. Take care to not set a too large number.
    - historian/limit_number_of_chunks_before_using_solr_partition : this option provides some performance tuning. Take care to not set a too large number.
    - historian/api/grafana/search/default_size : this option sets the maximum number of metrics to be returned, by default, for the endpoint Search.
    - historian/schema_version : The schema version to use. You should avoid changing this value by hand. It will changed by the system during a rolling update typically.
  - solr conf
    - historian/solr/connection_timeout : the connection timeout in milliseconds to the Solr server.
    - historian/solr/socket_timeout : the connection timeout in milliseconds for all reading sockets on Solr.
    - historian/solr/stream_url : the URL of the Solr collection to use for the stream API of Solr. We recommend to create a dedicated collection (with sufficient resources);

- historian/solr/chunk_collection : the name of the collection where time series are to be stored.
- historian/solr/annotation_collection : the name of the collection where annotations are to be stored.
- historian/solr/sleep_milli_between_connection_attempt : the number of milliseconds to wait between two ping attempts to the Solr server when starting the historian.
- historian/solr/number_of_connection_attempt : the number of attempts in testing connectivity to the Solr server when starting the historian.
- historian/solr/use_zookeeper : in case you are using Solr cloud (with or without a zookeeper server or cluster)
  - option if using zookeeper
    - historian/solr/zookeeper_urls : a list of at list one zookeeper server (ex: ["zookeeper1:2181"]).
    - historian/solr/zookeeper_chroot : the path to the root zookeeper that contains the Solr data. Do not enter or use null if there is no chroot (see the zookeeper documentation).
  - option if zookeeper is not used
    - historian/solr/urls : the http URLs to query Solr. For example ["http://server1:8983/solr", "http://server2:8983/solr"].

Generation of a configuration file, according to entered information, at installation.

Data management

Vous pouvez consulter notre documentation sur l'api REST 'rest-api.pdf' pour connaitre les détails de chaque fonction de l'API (endpoint). Maintenant que nous avons installé l'historian. Vous pouvez jouer avec des données, il y a plusieurs manières d'interagir avec l'historian selon votre culture et vos besoins.

L'historian ne contient initialement aucune données. Il existe plusieurs moyens d'injecter des données (voir le guide sur l'api rest), dans ce guide nous allons utiliser l'import de données à partir de fichiers csv.

Importing Data from CSV files with the REST API

TODO

Requêter des données avec l'api REST

```
curl --location --request POST 'http://localhost:8080/api/grafana/query' \
--header 'Content-Type: application/json' \
--data-raw '{
  "panelId": 1,
  "range": {
    "from": "2019-03-01T00:00:00.000Z",
    "to": "2020-03-01T23:59:59.000Z"
  },
  "interval": "30s",
  "intervalMs": 30000,
  "targets": [
    {
      "target": "\"ack\"",
      "type": "timeserie"
    }
  ],
  "format": "json",
  "maxDataPoints": 550
}'
```

Get all metrics names

```
curl --location --request POST 'http://localhost:8080/api/grafana/search' \
--header 'Content-Type: application/json' \
--data-raw '{
    "target": "*"
}'

# ["ack", ... ,"messages","cpu"]
```

Get some measures points within a time range

```
curl --location --request POST 'http://localhost:8080/api/grafana/query' \
--header 'Content-Type: application/json' \
--data-raw '{
     "range": {
          "from": "2019-11-25T23:59:59.999Z",
          "to": "2019-11-30T23:59:59.999Z"
      },
      "targets": [
        { "target": "ack" }
      ]
}'
```

Use Spark to get data

Apache Spark is an Open Source framework designed to process hudge datasets in parallel on computing clusters. Hurence Data Historian is highly integrated with Spark so that you can handle dataset interactions in both ways (input and output) through a simple API.

The following commands show you how to take a CSV dataset from HDFS or local filesystem, load it as a HDH

./_setup_historian/spark-2.3.4-bin-hadoop2.7/bin/spark-shell --jars assembly/target/historian-1.3.4-SNAPSHOT/historian-1.3.4-SNAPSHOT//lib/loader-1.3.4-SNAPSHOT.jar,assembly/target/historian-1.3.4-SNAPSHOT/historian-1.3.4-SNAPSHOT/lib/spark-solr-3.6.6-shaded.jar

```
import com.hurence.historian.model.ChunkRecordV0
import com.hurence.historian.spark.ml.Chunkyfier
import com.hurence.historian.spark.sql
import com.hurence.historian.spark.sql.functions._
import com.hurence.historian.spark.sql.reader.{MeasuresReaderType, ReaderFactory}
import com.hurence.historian.spark.sql.writer.{WriterFactory, WriterType}
import com.lucidworks.spark.util.SolrSupport
import org.apache.commons.cli.{DefaultParser, Option, Options}
import org.apache.spark.sql.SparkSession
import org.slf4j.LoggerFactory

 val filePath =
"/Users/tom/Documents/workspace/historian/loader/src/test/resources/it-data-
4metrics.csv.gz"
 val reader = ReaderFactory.getMeasuresReader(MeasuresReaderType.GENERIC_CSV)
    val measuresDS = reader.read(sql.Options(
      filePath,
      Map(
        "inferSchema" -> "true",
        "delimiter" -> ",",
        "header" -> "true",
        "nameField" -> "metric_name",
        "timestampField" -> "timestamp",
        "timestampDateFormat" -> "s",
        "valueField" -> "value",
        "tagsFields" -> "metric_id,warn,crit"
      )))



val chunkyfier = new Chunkyfier().setGroupByCols(Array(  "name", "tags.metric_id"))
val chunksDS = chunkyfier.transform(measuresDS).as[ChunkRecordV0]


val writer = WriterFactory.getChunksWriter(WriterType.SOLR)
    writer.write(sql.Options("historian", Map(
      "zkhost" -> "localhost:9983",
      "collection" -> "historian",
      "tag_names" -> "metric_id,warn,crit"
    )), chunksDS)
```

Realtime data ingestion with logisland

Hurence's Open Source software for stream processing (therefore real-time data processing), LogIsland, allows you to inject data "on the fly", especially those that you could "push" in a Mqtt

message bus or Kafka.

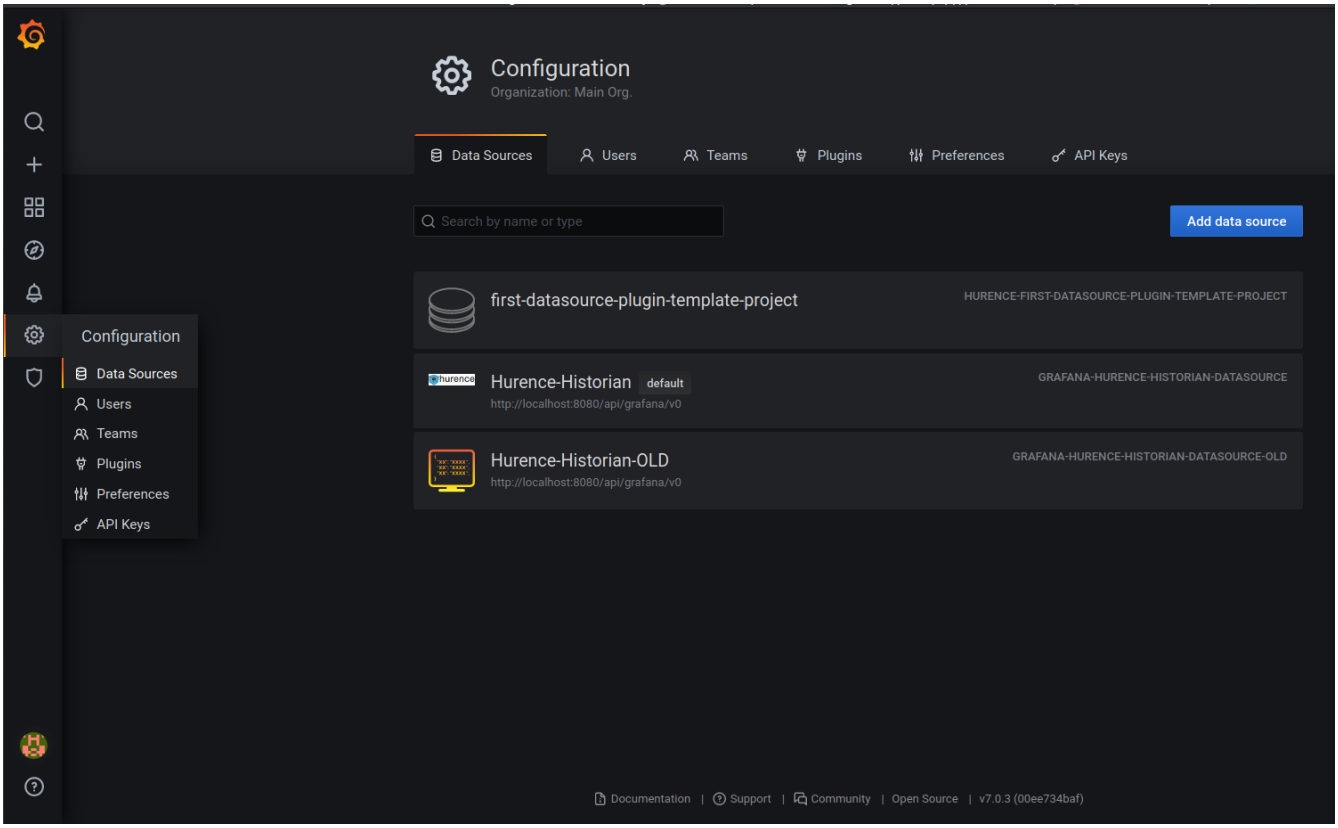
Hurence Data Historian is therefore able to process sensors data and to store and graph it in real-time. Hurence also has some OPC connectors to get sensors data from factory equipements.

To set up a real-time injection chain, the best is to contact Hurence (contact@hurence.com) for a little support because it becomes real Big Data in real time.

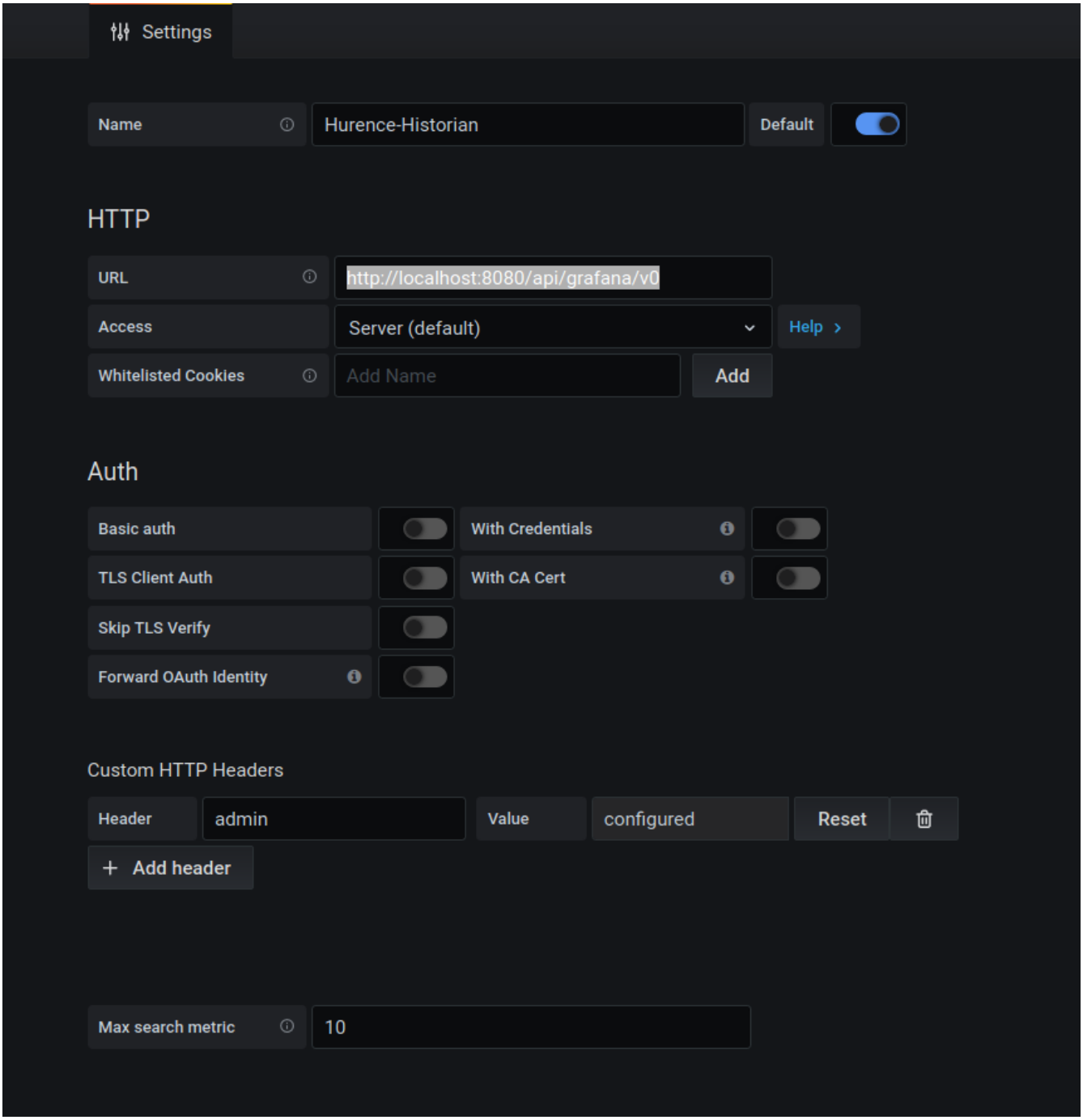
Data visualization

To create dashboards you have to go to the graphical interface of grafana (http://localhost:3000/) (localhost being replaced by a machine name if you are not in standalone installation).

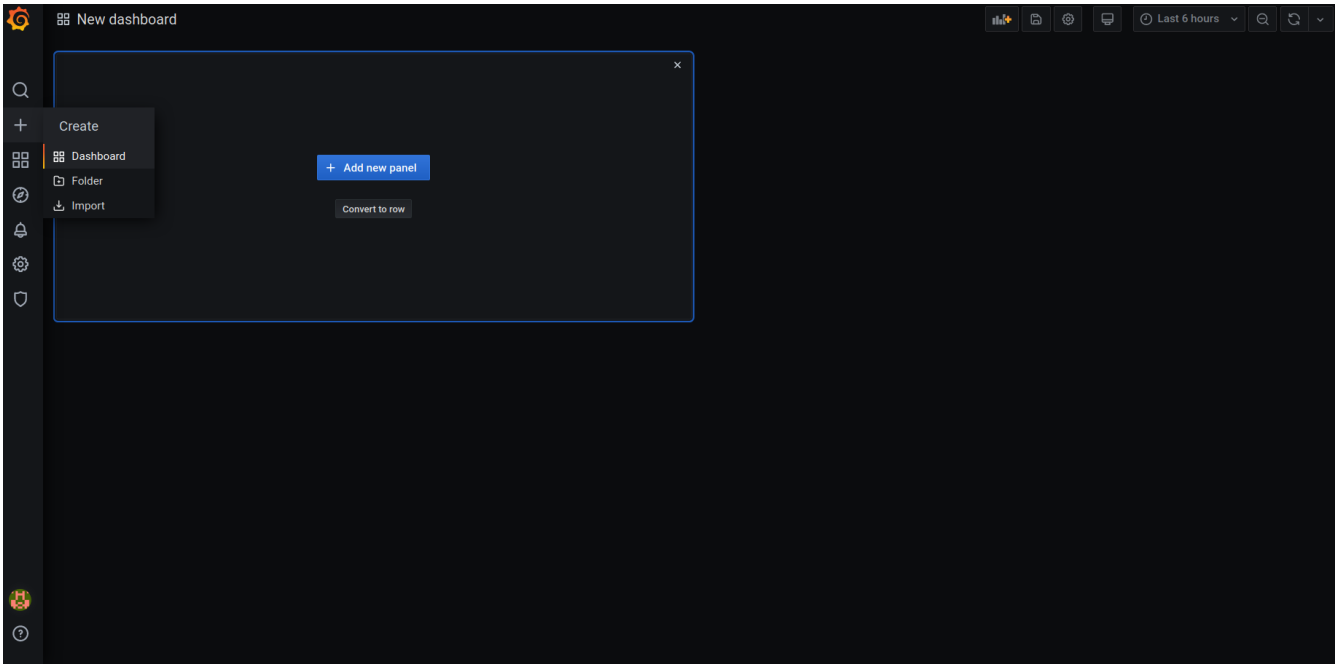
Then go to the datasources menu:



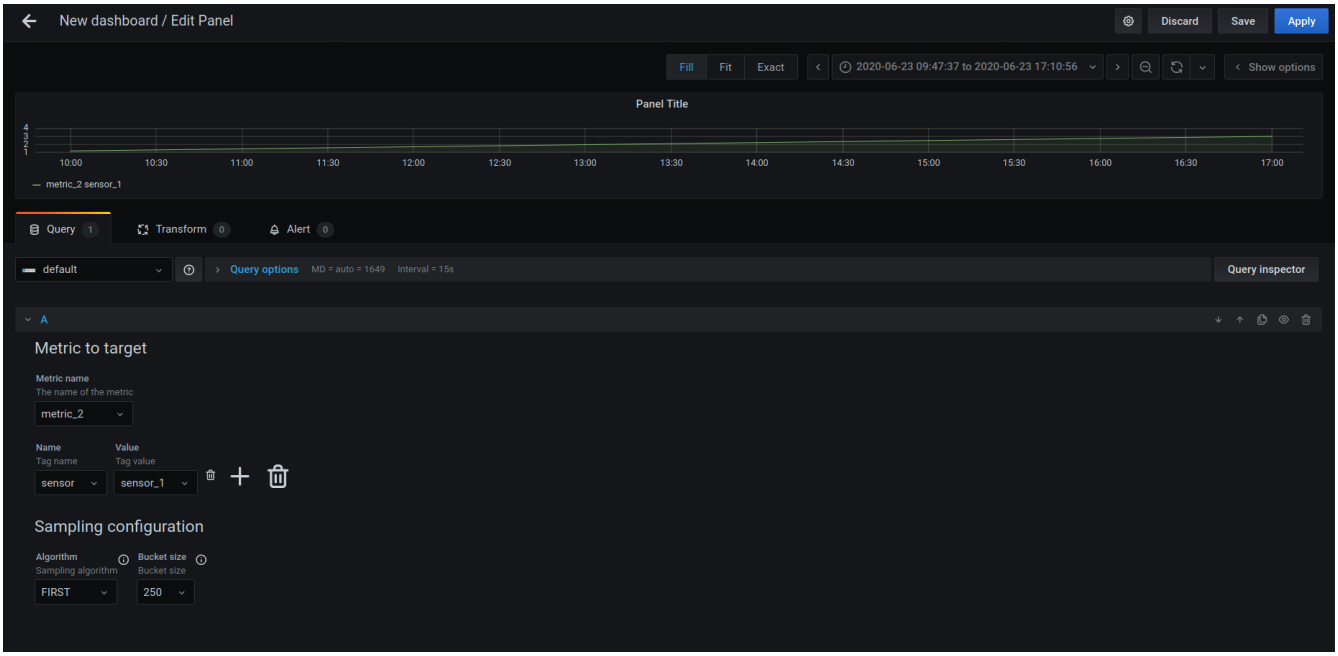
Look for the "Hurence-Historian" datasource and select it. You just have to enter the URL http://localhost:8080/api/grafana/v0 in the case of standalone installation.



Test the connectivity by clicking on the "Save & Test" button at the bottom of the page. Then create a new dashboard.



Add the graphs you want, a small example below:



A "query" consists of entering a metric name, and pairs of name / tag value tags as well as options for the sampling algorithm.

note

the sampling options of the first request are used for all the others.

note

if you don’t have data injected, follow the tutorial to inject data.

note

if you did not add a name tag during the installation you will not be able to use tags. It is always possible to add tags manually after installation by adding a field in the diagram.

Stopping and deleting data

This section teaches you how to stop the services and destroy the data if necessary (after testing you will want to feed with your real data).

Stopping your Solr instances

This section teaches you how to stop Solr intances. Please note this command will switch off Solr (this could impact other services using Solr).

```
cd $SOLR_HOME
bin/solr stop -all
```