Manuel utilisateur

(C)Hurence

Version v1.0, 18.02.2020: First draft

Table des matières

Introduction

Ce guide aidera l'utilisateur du data historian dans la compréhension de la plateforme, de son installation, de son utilisation. Le data historian ayant été conçu dans l'optique d'un déploiement sur une infrastructure Big Data, qui sont par nature des environnements complexes à mettre en oeuvre, nous n'aborderons dans ce guide que les déploiements sur une seule machine et de manière progressive. Pour un déploiement sur une infrastructure Big Data multi-serveurs et sa sécurisation, il faudra se référer à d'autres documentations ou contacter Hurence à 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 arggregation 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As you can see from a Measure points to a Chunk of Measures, the `timestamp` field has been replaced by a `start` and `stop` interval and the `value` is now a base64 encoded string named `chunk`.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

Cette section décrit une installation simple du data historian sur un seul serveur. Le Data Historian étant conçu pour le Big Data, il est possible de réaliser une installation avec des possibilités de stockage en très grandes volumétries et avec des performances incomparables sur plusieurs racks de serveurs. Ce type d'installation n'est pas décrite ici par simplification car elle concerne des experts en urbanisation d'infrastructures Big Data.

NOTE	Hurence propose de l'accompagnement pour installer le data historian pour des volumétries importantes.

Pre-requis pour une installation mono serveur en production

La configuration minimale du serveur pour une installation mono serveur (en production) du data historian est:

- un OS CentOS ou Redhat 7.x
- 32 Gigabits de RAM
- 32 vcores de CPU
- 2 Tera octets de disque
- Java 8

Il vous faudra également les logiciels ci-dessous, si vous ne les avez pas installés on vous fournira une aide.

- solr 8.2.0
- grafana (une version récente), l'historian a été tester avec la version TODO

Installation de Apache SolR

Vous pouvez installer solr vous-même (recommandé pour un environnement de production) ou bien laisser le script d'installation installer une version standalone de solr. Si vous choisissez de passer par le script d'installation (pour tester), vous pouvez sauter cette section.

Apache SolR est la base de donnée utilisé par l'historian, elle peut être remplacé par un autre moteur de recherche.

Vous pouvez télécharger la version 8.2.0 de solr sur le lien suivant : https://archive.apache.org/dist/lucene/solr/8.2.0/solr-8.2.0.tgz
Décompressez l'archive dans un répertoire de votre choix puis entrez dans le répertoire générer par la décompression, on appellera par la suite ce répertoire '$SOLR_HOME'.

Pour l'exemple de l'installation en mode standalone nous allons lancer deux cores solr localement dans le répertoire `$SOLR_HOME/data`.

```
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
```

Vérifiez que votre instance solr fonctionne correctement en allant sur l'interface graphique à l'adresse suivante : http://localhost:8983/solr/#/~cloud

Installation de Grafana

Vous pouvez installer grafana vous même (recommandé pour un environnement de production) ou bien laisser le script d'installation installer une version standalone de grafana. Si vous choisissez de passer par le script d'installation (pour tester), vous pouvez sauter cette section.

Installez Grafana pour votre platform comme décrit ici : `https://grafana.com/docs/grafana/latest/installation/requirements/`

Installation Plugin Grafana pour utiliser le data historian Hurence

TODO

Install Json Simple Datasource as described here :

https://grafana.com/grafana/plugins/grafana-simple-json-datasource

To install this plugin using the grafana-cli tool:

```
sudo grafana-cli plugins install grafana-simple-json-datasource
sudo service grafana-server restart
```

Installation de l'historian

Hurence Data Historian est composé de scripts et fichiers binaires qui permettent de travailler avec les timeseries et les chunks. Téléchargez la dernière version de l'historian à l'adresse suivante https://github.com/Hurence/historian/releases/download/v1.3.4/historian-1.3.4-SNAPSHOT.tgz. Décompressez l'archive et entrez dedans, ensuite commencez l'installation en tapant la commande suivante :

```
sudo ./bin/install.sh
```

Il sera alors demander à l'utilisateur plusieurs informations, si vous ne savez pas quoi mettre laissez les valeurs par défaut, dans ce cas l'historian téléchargera et installera solr 8.2 ainsi que grafana localement.

Attention cependant au chemin ou vous installez l'historian en mode solr standalone et/ou en mode grafana standalone. En effet les données seront alors stockées dans le home de l'historian, il est donc nécessaire que l'espace disque soit adapté. Voici un exemple :

TODO remplacer par un screenshot

```
Where do you want to install Hurence Data Historian ? [/opt/hdh]
>
Will you use an existing solr install or let the historian install an embedded solr
(version 8.2.0 rquired) ? Standalone(S)/Existing(E) [S]
> S
Which name to use for the solr collection which will be storing timeseries ? [TODO]
>
do you want to install an embedded grafana (version TODO rquired) ? Yes(Y)/No(N) [Y]
>
```

Dans le cas ou vous voulez utiliser des intallation de solr et/ou grafana existante voilà un exemple d'installation :

```
Where do you want to install Hurence Data Historian ? [/opt/hdh]
>
Will you use an existing solr install or let the historian install an embedded solr
(version 8.2.0 rquired) ? Standalone(S)/Existing(E) [S]
> E
What is the path to the solr cluster ? We will use the solr REST api to create
collection.
> http://mysolrhost:myport/solr
Which name to use for the solr collection which will be storing timeseries ? [TODO]
>
do you want to install an embedded grafana (version TODO rquired) ? Yes(Y)/No(N) [Y]
> N
```

Dans la suite, on appellera le chemin indiqué pour l'installation de l'historian '$HDH_HOME'.

A l'issue de ce script, si vous décidez d'utiliser tous les paramètres par défaut vous aurez :

- Un serveur solr 8.2 installé dans $HDH_HOME/solr-8.2.0
- Le script à démarrer ce serveur solr, vous pouvez vérifier cela à l'adresse suivante : http://localhost:8983/solr/cloud](http://localhost:8983/solr/ /cloud). Vous pouvez regarder la documentation solr pour interagir avec solr.
- Un serveur grafana version-TODO installé dans $HDH_HOME/grafana
- Le plugin [datasource grafana de l'historian](https://github.com/Hurence/grafana-historian-datasource) installer sur ce serveur grafana.
- Le serveur grafana a du être lancé par le script, vous pouvez vérifier à l'adresse suivante : http://localhost:3000/](http://localhost:3000/). Vous pouvez regarder la documentation solr pour interagir avec grafana.

Voici la structure de $HDH_HOME a l'issue de l'installation par défaut : * $HDH_HOME/data : TODO * $HDH_HOME/solr-8.2.0 : TODO * $HDH_HOME/bin/historian-server.sh : Permet de lancer et arrêter l'api REST de l'historian. * $HDH_HOME/conf/log4j.properties : Fichier pour contrôler le niveau des logs en mode production (défaut). * $HDH_HOME/conf/log4j-debug.properties : Fichier pour contrôler le niveau des logs en mode debug. * $HDH_HOME/conf/historian-server-conf.json : Le fichier de configuration du serveur fournissant l'api rest de l'historian.

Le script $HDH_HOME/bin/historian-server.sh sert à lancer/arrêter l'api rest de l'historian.

Pour lancer l'historian taper la commande suivante :

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

Pour arrêter l'historian taper la commande suivante :

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

Attention ces commandes n'affectent ni grafana ni solr qui sont des services indépendants.

Description du fichier de configuration de l'historian

Voici le fichier de configuration par défaut, il contient toutes les informations possibles, certaines ne sont pas obligatoire :

```
{
  "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,
  },
  "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 : Le nombre d'instances de verticles à déployer pour répondre aux appelles http des clients. Un verticle est capable de gérer un grand nombre de requête (au moins 1000, voir la documentation de vertx pour plus d'information).
  - historian.verticles.instance.number : Le nombre d'instances de verticles à déployer pour le service de l'historian qui s'occupe du sampling et des interactions avec le backend. C'est ce paramètre qui va être clé, il y a de grande chance que ce soit ce paramètre qu'il faille augmenter en cas de problème de performances.
- Http server conf :

    - http_server/host : le nom du serveur http à déployer.
    - http_server/port : le port sur lequelle déployé l'api rest.
    - http_server/historian.address : le nom du service historian vertx déployé. Ne pas modifier sauf si vous hébergez d'autres services vertx et que vous savez ce que vous faites
    - http_server/max_data_points_allowed_for_ExportCsv : Ici vous pouvez modifier le maximum de points que l'historian accepte de retourner lorsqu'un client utiliser l'api rest d'export dans le format csv. Attention de ne pas choisir un maximum trop grand car il faut que cela tienne en mémoire. Si vous avez besoin d'un gros export il vous faudra utiliser un outil comme spark.
- Historian service conf :
    - general conf
        - historian/address : le nom du service historian vertx déployé. Ne pas modifier sauf si vous hébergez d'autres services vertx et que vous savez ce que vous faites. Doit être identique à la valeur de 'http_server/historian.address'.
        - historian/limit_number_of_point_before_using_pre_agg : Une option pour optimiser les performances. Attention à ne pas mettre un nombre trop grand.
        - historian/limit_number_of_chunks_before_using_solr_partition : Une option pour optimiser les performances. Attention à ne pas mettre un nombre trop grand.
        - historian/api/grafana/search/default_size : Une option pour modifier le nombre maximum de nom de métrique à retourner par défaut pour l'endpoint search.
        - historian/schema_version : La version du schéma a utiliser. (Attention ne pas modifier cette valeur manuellement !)
    - solr conf
        - historian/solr/connection_timeout : Le timeout lors de la connection au serveur Solr en millisecondes.
        - historian/solr/socket_timeout : Le timeout pour tous les socket de lecture avec Solr en millisecondes.
        - historian/solr/stream_url : l'url de la collection solr a utiliser pour l'api stream de solr. Il est recommandé de créer une collection dédié (avec les ressources suffisantes).
        - historian/solr/chunk_collection : Le nom de la collection ou sont stockés les timeseries.
        - historian/solr/annotation_collection : Le nom de la collection ou sont stockés les annotations.
        - historian/solr/sleep_milli_between_connection_attempt : Le nombre de millisecondes à attendre entre chaque tentatives de ping du serveur solr au démarrage de l'historian.
        - historian/solr/number_of_connection_attempt : Le nombre de tentatives pour tester la connectivité au serveur solr au démarrage de l'historian.
        - historian/solr/use_zookeeper : If you use solr cloud (using a zookeeper server) or not.
            - option si utilisation de zookeeper
                - historian/solr/zookeeper_urls : une liste d'au moins un serveur zookeeper (ex: ["zookeeper1:2181"]).

        - historian/solr/zookeeper_chroot : Le chemin root zookeeper qui contient les données solr. Ne pas renseigné ou utiliser null si il n y a pas de chroot (voir documentation zookeeper).
    - option si zookeeper n'est pas utilisé
        - historian/solr/urls : Les urls http pour faire des requêtes à solr. Par exemple ["http://server1:8983/solr", "http://server2:8983/solr"].

TODO éventuellement générer un fichier de configuration pendant le script d'install selon les informations renseignées ?

Install Apache Spark

Cette étape n'est pas obligatoire si vous ne voulez pas traiter vos dpnnées avec le framework spark pour effectuer des traitement sur des grosses volumétries. Sinon pour installer spark vous pouvez téléchargez cette archive : https://archive.apache.org/dist/spark/spark-2.3.4/spark-2.3.4-bin-without-hadoop.tgz

Les commandes suivantes vous permettront d'avoir une installation locale.

```
# 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/
```

Tear down and cleanup

This section shows how to stop the services and cleanup data if needed.

Stop your SolR instances

when you're done you can stop your SolR cores. Attention cette commande va éteindre Solr (cela pourrait impacter d'autres service utilisant solr).

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

Si vous ête en mode standalone pour solr, vous pouvez effacer vos données en supprimer les fichiers suivants

```
rm -r $HDH_HOME/data/solr/node1 $HDH_HOME/data/solr/node2
```

TODO see what is needed to rebootstrap. * Peut être prévoir un script de désinstall. * Un script pour reset les données solr.

don't forget then to bootstrap your setup again as described previously.

Data management

Vous pouvez consulter notre documentation sur l'api REST 'rest-api.pdf' pour connaitre les détails de chaque endpoint. Maintenant que nous avons installer 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.

Import de donnée à partir de fichiers csv avec l'api REST

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" }
      ]
}'
```

Retrieve data from REST API

TODO

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)
```

Visualize data through grafana

Realtime data ingestion with logisland