CCP Enriching Telemetry Events 1.8.0

Enriching Telemetry Events

Date of publish: 2017-11-06

CLOUDERA

https://docs.cloudera.com/

Legal Notice

The documentation is and contains Cloudera proprietary information protected by copyright and other intellectual property rights. No license under copyright or any other intellectual property right is granted herein.

Copyright information for Cloudera software may be found within the documentation accompanying each component in a particular release.

Cloudera software includes software from various open source or other third party projects, and may be released under the Apache Software License 2.0 (“ASLv2”), the Affero General Public License version 3 (AGPLv3), or other license terms. Other software included may be released under the terms of alternative open source licenses. Please review the license and notice files accompanying the software for additional licensing information.

Please visit the Cloudera software product page for more information on Cloudera software. For more information on Cloudera support services, please visit either the Support or Sales page. Feel free to contact us directly to discuss your specific needs.

Cloudera reserves the right to change any products at any time, and without notice. Cloudera assumes no responsibility nor liability arising from the use of products, except as expressly agreed to in writing by Cloudera.

Cloudera, Cloudera Altus, HUE, Impala, Cloudera Impala, and other Cloudera marks are registered or unregistered trademarks in the United States and other countries. All other trademarks are the property of their respective owners.

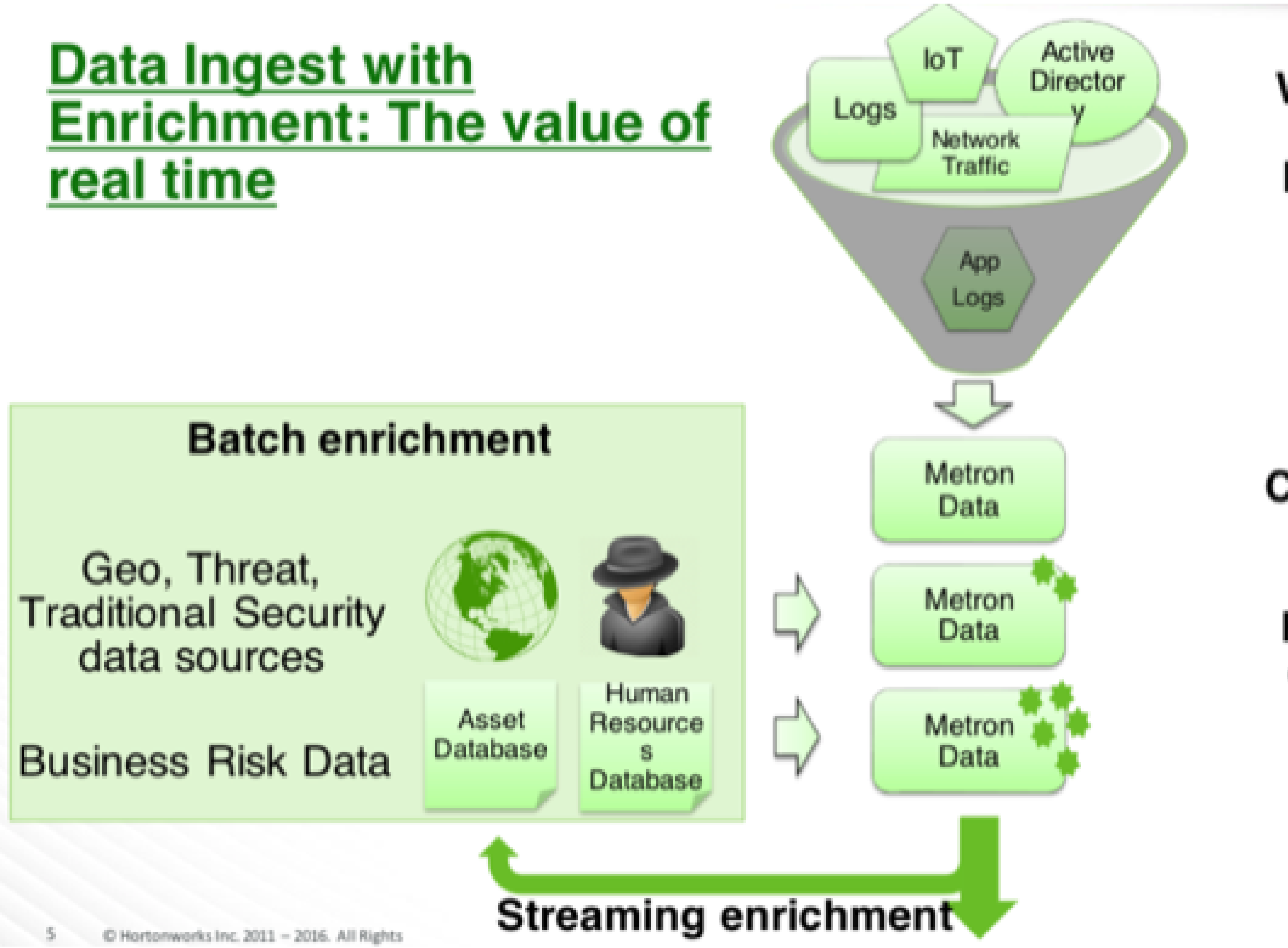
Disclaimer: EXCEPT AS EXPRESSLY PROVIDED IN A WRITTEN AGREEMENT WITH CLOUDERA, CLOUDERA DOES NOT MAKE NOR GIVE ANY REPRESENTATION, WARRANTY, NOR COVENANT OF ANY KIND, WHETHER EXPRESS OR IMPLIED, IN CONNECTION WITH CLOUDERA TECHNOLOGY OR RELATED SUPPORT PROVIDED IN CONNECTION THEREWITH. CLOUDERA DOES NOT WARRANT THAT CLOUDERA PRODUCTS NOR SOFTWARE WILL OPERATE UNINTERRUPTED NOR THAT IT WILL BE FREE FROM DEFECTS NOR ERRORS, THAT IT WILL PROTECT YOUR DATA FROM LOSS, CORRUPTION NOR UNAVAILABILITY, NOR THAT IT WILL MEET ALL OF CUSTOMER’S BUSINESS REQUIREMENTS. WITHOUT LIMITING THE FOREGOING, AND TO THE MAXIMUM EXTENT PERMITTED BY APPLICABLE LAW, CLOUDERA EXPRESSLY DISCLAIMS ANY AND ALL IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO IMPLIED WARRANTIES OF MERCHANTABILITY, QUALITY, NON-INFRINGEMENT, TITLE, AND FITNESS FOR A PARTICULAR PURPOSE AND ANY REPRESENTATION, WARRANTY, OR COVENANT BASED ON COURSE OF DEALING OR USAGE IN TRADE.

Contents

Enrichment Overview

After you have parsed and normalized the raw security telemetry events, the next step is to enrich the data elements of the normalized event.

Enrichments add external data from data stores (such as HBase). Cloudera Cybersecurity Platform (CCP) uses a combination of HBase, Storm, and the telemetry messages in json format to enrich the data in real time to make it relevant and consumable. For example, the GEO enrichment provides latitude and longitude coordinates plus the city, state, and country for an external IP address. You can use this enriched information immediately rather than needing to hunt in different silos for the relevant information.



Data Ingest with Enrichment: The value of real time

CCP provides a separate HBase table named enrichment_list that holds the list of enrichment types. It is dynamically populated by listening to inserts made on the "enrichment" table and helps to provide an accurate list of all available enrichment types.

CCP provides two types of enrichment:

- Telemetry events
- Threat intelligence information

The telemetry data sources for which CCP includes parsers (for example, Bro, Snort, and YAF) already include the following enrichment topologies that activate when you start the data sources in CCP:

- Asset
- GeoIP
- User

You can also add enrichment sources for your own parsers to suit your needs. For example, you might want to use one or more of the following:

Transformation	Extracts data from existing fields. Brings in profile or machine learning results.
Geocoding	Adds information about city, country, postal code, latitude, and longitude for IP addresses.
Organization Classification	Provides information about host, user, and any other assets derived from organizational data sources such as CMDB or employee databases.
Whois	Provides ownership information of the web domain.

One of the advantages of the enrichment topology is that it groups messages by HBase key. Whenever you execute a Stellar function, you can add a caching layer, thus decreasing the need to call HBase for every event.

Prior to enabling an enrichment capability within CCP, the enrichment store (which for CCP is primarily HBase) must be loaded with enrichment data. The dataload utilities convert raw data sources to a primitive key (type, indicator) and value and place it in HBase. You can load enrichment data from the local file system or HDFS, or you can use the parser framework to stream data into the enrichment store. The enrichment loader transforms the enrichment into a JSON format that CCP can use. Additionally, the loading framework can detect and remove old data from the enrichment store automatically.

CCP supports three types of enrichment loaders:

- Bulk load from HDFS via MapReduce
- Taxii Loader
- Flat File ingestion

After you load the stores, you can incorporate into the enrichment topology an enrichment bolt to a specific field or tag within a Metron message. The bolt can detect when it can enrich a field and then take an enrichment from the enrichment store and tag the message with it. The enrichment is then stored within the bolt's in-memory cache. CCP uses the underlying Storm routing capabilities to ensure that similar enrichment values are sent to the appropriate bolts that already have these values cached in-memory.

CCP supports two types of configurations: global and sensor specific. The sensor specific configuration configures the individual enrichments and threat intelligence enrichments for a given sensor type (for example, squid). This section describes sensor specific configurations.

Setting Up Enrichment Configurations

You can use the enrichment topology to enhance messages with external data and manage threat intelligence data.

The enrichment topology is a topology dedicated to performing the following:

- Taking the data from the parsing topologies normalized into the Metron data format (for example, a JSON Map structure with original_messageand timestamp.
- Enriching messages with external data from data stores (for example, hbase) by adding new fields based on existing fields in the messages.
- Marking messages as threats based on data in external data stores.
- Marking threat alerts with a numeric triage level based on a set of Stellar rules.

The configuration for the `enrichment` topology, the topology primarily responsible for enrichment and threat intelligence enrichment, is defined by JSON documents stored in ZooKeeper.

There are two types of configurations, global and sensor specific.

Sensor Configuration

You can use the sensor-specific configuration to configure the individual enrichments and threat intelligence enrichments for a given sensor type (for example, Snort). The sensor configuration format is a JSON object stored in Apache ZooKeeper.

The sensor enrichment configuration uses the following fields:

fieldToTypeMap

In the case of a simple HBase enrichment (a key/value lookup), the mapping between fields and the enrichment types associated with those fields must be known. This enrichment type is used as part of the HBase key. Note: applies to hbaseEnrichment only. | `"fieldToTypeMap" : { "ip_src_addr" : [ "asset_enrichment" ] }` |

fieldMap

The map of enrichment bolts names to configuration handlers which know how to divide the message. The simplest of which is just a list of fields. More complex examples would be the Stellar enrichment which provides tellar statements. Each field listed in the array arg is sent to the enrichment referenced in the key. Cardinality of fields to enrichments is many-to-many. | `"fieldMap": {"hbaseEnrichment": ["ip_src_addr","ip_dst_addr"]}` |

config

The general configuration for the enrichment.

The `config` map is intended to house enrichment specific configuration. For instance, for hbaseEnrichment, the mappings between the enrichment types to the column families is specified.

The fieldMap contents are of interest because they contain the routing and configuration information for the enrichments. When we say 'routing', we mean how the messages get split up and sent to the enrichment adapter bolts.

The simplest, by far, is just providing a simple list as in

```
"fieldMap": {
      "geo": [
        "ip_src_addr",
        "ip_dst_addr"
      ],
      "host": [
        "ip_src_addr",
        "ip_dst_addr"
      ],
      "hbaseEnrichment": [
        "ip_src_addr",
        "ip_dst_addr"
      ]
      }
```

Based on this sample config, both ip_src_addr and ip_dst_addr will go to the `geo`, `host`, and `hbaseEnrichment` adapter bolts.

Transform Data Source Parser Information

After you create a parser, you can use the CCP Management UI to transform the data source data to provide more relevant and helpful information. Transformation creates a new event field based on an existing field value. For example, you can choose to transform a url to provide the domain name of the outbound connection or the IP address.

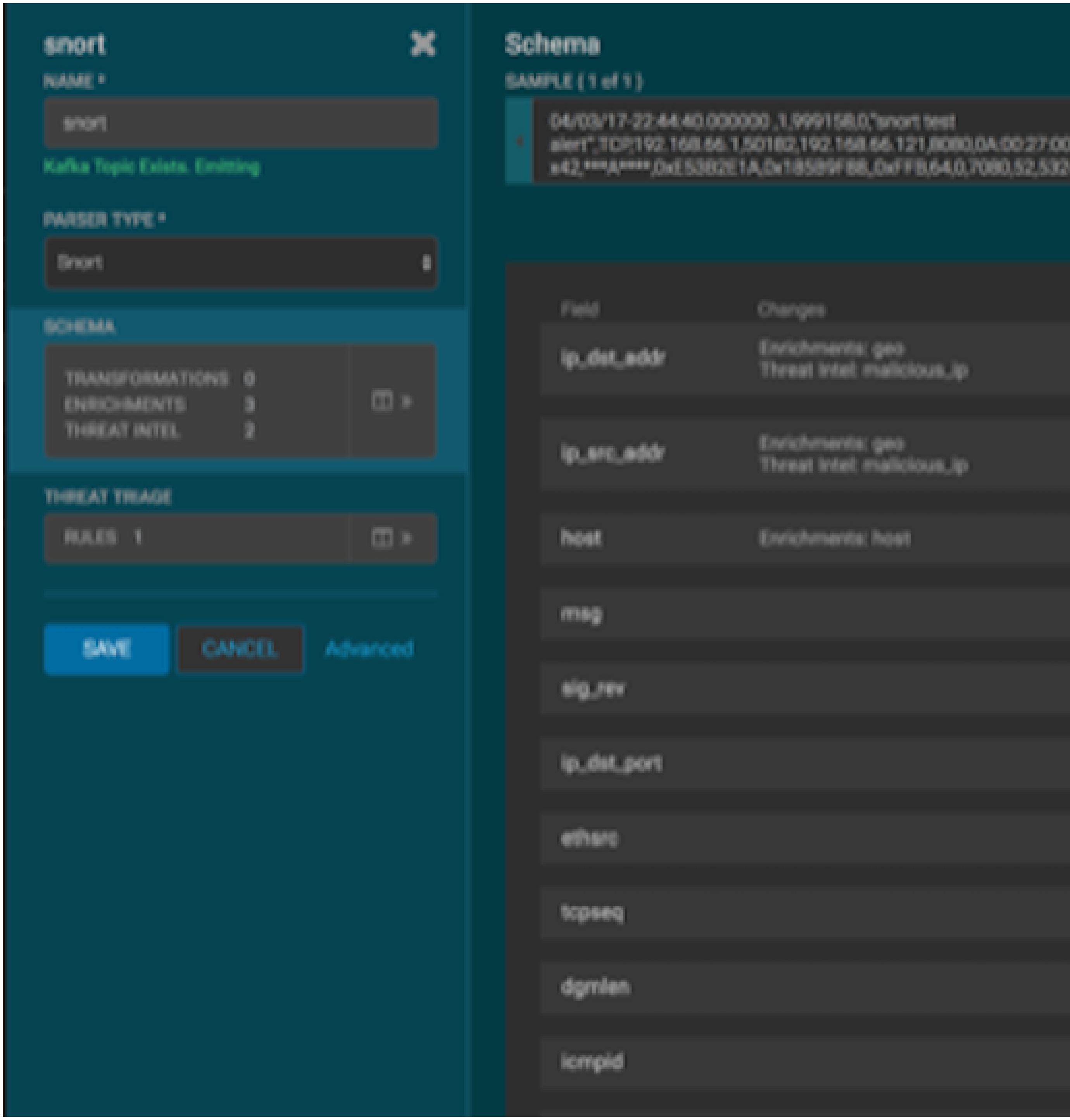
Procedure

1. From the list of sensors in the Management user interface main window, select your new sensor.
2. Click the pencil icon in the toolbar.

   The Management UI displays the sensor panel for the new sensor.

   Note: Your sensor must be running and producing data before you can add transformation information.
3. In the Schema box, click ◫ » (expand window).

   The Management UI populates the panel with message, field, and value information.



snort

NAME *

snort

Kafka Topic Exists. Emitting

PARSER TYPE *

Snort

SCHEMA

TRANSFORMATIONS	0
ENRICHMENTS	3
THREAT INTEL	2

THREAT TRIAGE

RULES 1

Schema

SAMPLE (1 of 1)

```
04/03/17-22:44:40.000000 ,1,999158,0,"snort test
alert",TCP,192.168.66.1,50182,192.168.66.121,8080,0A:00:27:00
x42,***A****,0xE5382E1A,0x18589F88,0xFFB,64,0,7080,52,532
```

Field	Changes
ip_dst_addr	Enrichments: geo Threat Intel: malicious_ip
ip_src_addr	Enrichments: geo Threat Intel: malicious_ip
host	Enrichments: host
msg	
sig_rev	
ip_dst_port	
ethsrc	
tcpseq	
dgmlen	
icmpid	

The Sample field displays a parsed version of a sample message from the sensor. The Management UI tests your transformations against these parsed messages.

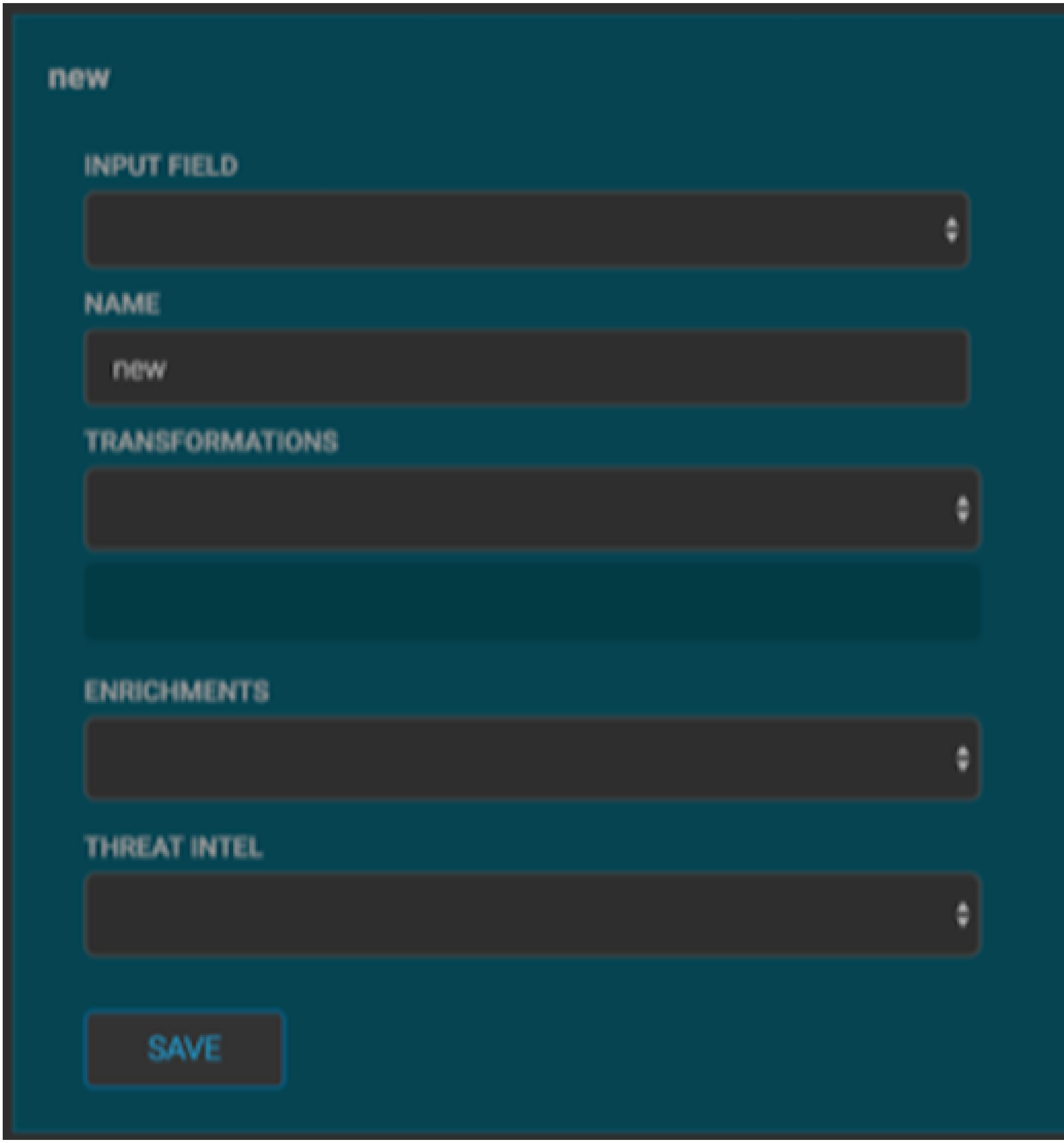
You can use the right and left arrows to view the parsed version of each sample.

Although you can apply transformations to an existing field, users typically create and transform a new field.

4. To add a new transformation, either click ✎ (edit) next to a field or click

(add) at the bottom of the **Schema** panel.

The following dialog box displays:



new

INPUT FIELD

NAME

new

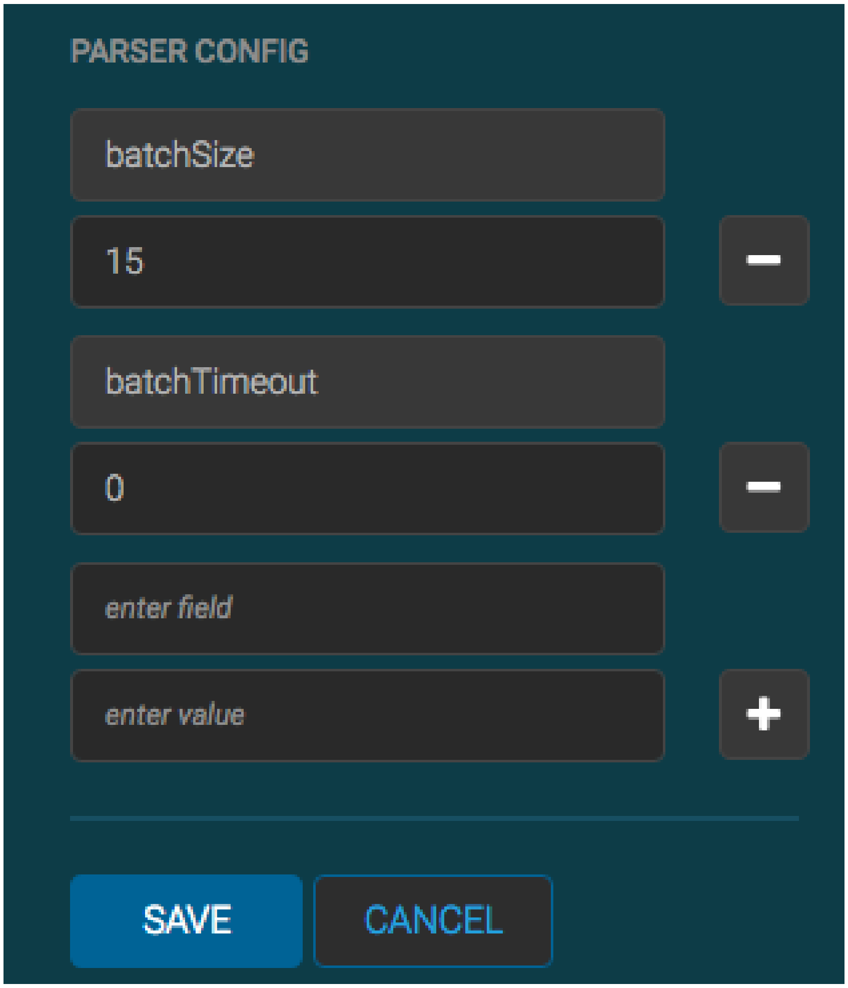
TRANSFORMATIONS

ENRICHMENTS

THREAT INTEL

5. From INPUT FIELD, select the field to transform, enter the name of the new field in the NAME field, and then choose a function with the appropriate parameters in the TRANSFORMATIONS box.
6. If you decide not to use the default values for the batchSize and batchTimeout properties, you can set their values.

   In the **Advanced** portion of the input panel, enter the property name (for example batchSize) and the value in the **PARSER CONFIG** fields.



PARSER CONFIG

batchSize

15

batchTimeout

0

7. Click **SAVE**.

   If you change your mind and want to remove a transformation, click the "x" next to the field.
8. Alternatively, you can add enrichment properties using the **RAW JSON** field in the **Advanced** section of the **Schema** panel.
9. You can also suppress fields with the transformation feature by clicking ⊘ (suppress icon).

   This icon prevents the field from being displayed, but it does not remove the field entirely.
10. Click **SAVE**.

Configure Geocoding

You can enrich your parser data by adding locations of any known external IP address. The IP address must exist in the geocoding data that you are loading into Cloudera Cybersecurity Platform (CCP).

Procedure

1. From the list of sensors in the Management user interface main window, select your new sensor.

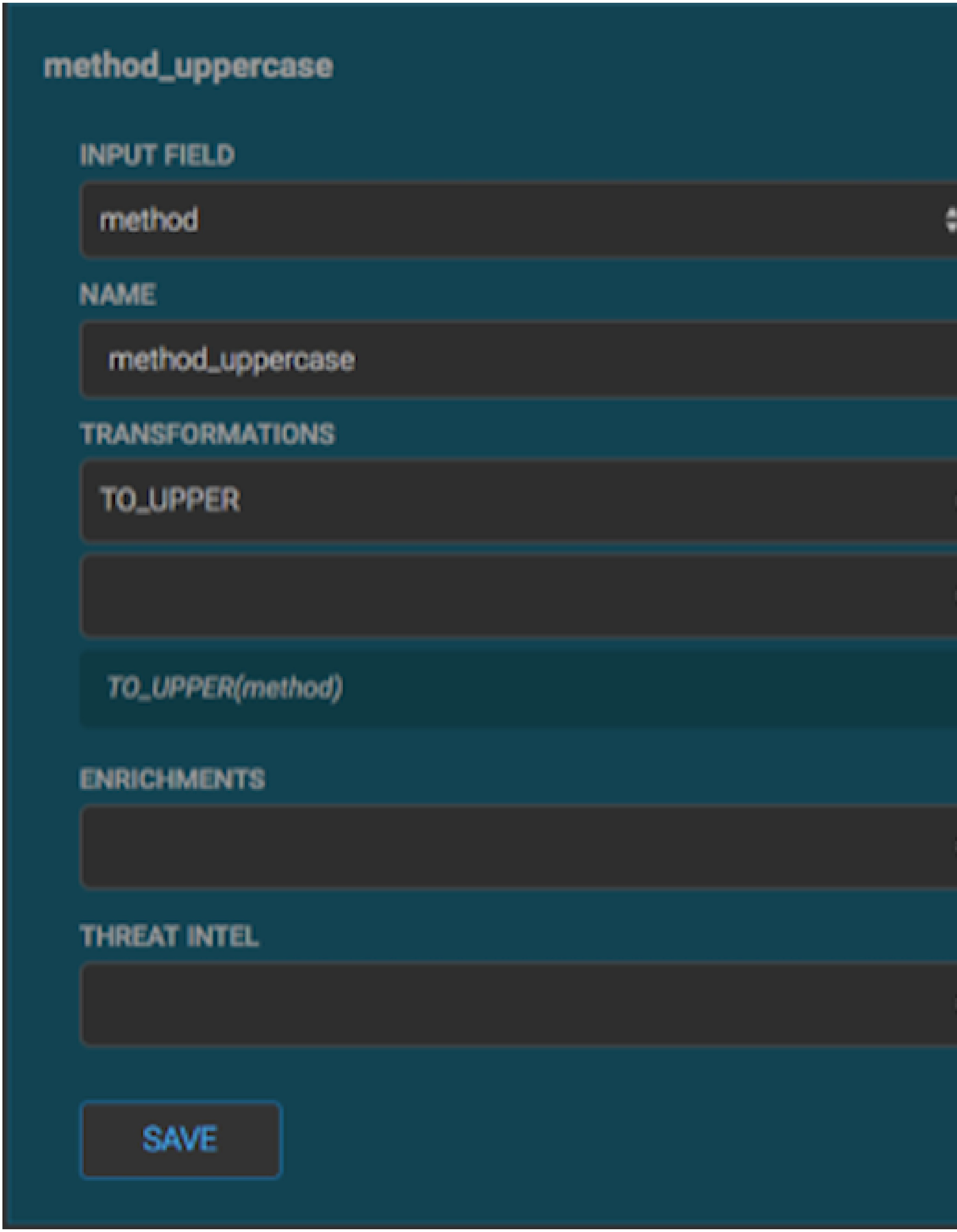
2. Click the pencil icon in the toolbar.

   The Management UI displays the sensor panel for the sensor.

   Note: Your sensor must be running and producing data before you can add geocoding information.

3. In the Schema box, click ◫ » (expand window).

   The Management UI displays a panel for the sensor.



method_uppercase

INPUT FIELD

method

NAME

method_uppercase

TRANSFORMATIONS

TO_UPPER

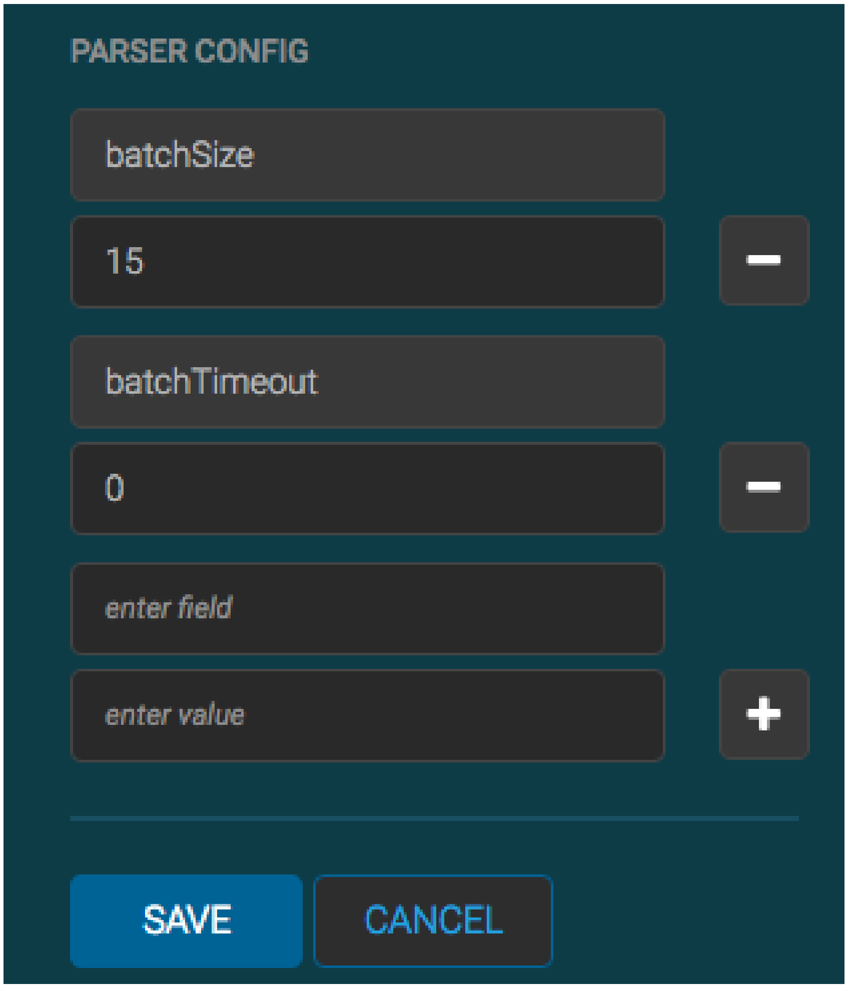
TO_UPPER(method)

ENRICHMENTS

THREAT INTEL

4. From INPUT FIELD, select the field to transform, enter the name of the new field in the NAME field, and then choose a function with the appropriate parameters in the TRANSFORMATIONS box.
5. From the **ENRICHMENTS** field, choose geo.
6. If you decide not to use the default values for the batchSize and batchTimeout properties, you can set their values.

   In the **Advanced** portion of the input panel, enter the property name (for example batchSize) and the value in the **PARSER CONFIG** fields.



PARSER CONFIG

batchSize

15

batchTimeout

0

7. Click **SAVE**.

   If you change your mind and want to remove a transformation, click the "x" next to the field.
8. You can also suppress fields with the transformation feature by clicking ⊘ (suppress icon).

   This icon prevents the field from being displayed, but it does not remove the field entirely.
9. Click **SAVE**.

Load Geocoding Information

You need to reload geocoding information periodically because the context will change over time.

Procedure

1. Ensure that Metron has set the location of the GeoIP database.

   a) From Ambari, display the Metron service.

   b) Enter geo in the search field.

Ambari displays the location under in the **GEOIP Load Defaults URL** field under **Adapter**.

2. To reload the geocoding information into HDFS, as run the following:

   ```
   cd /usr/hcp/current/metron
   sudo su hdfs -c "bin/geo_enrichment_load.sh -z localhost:2181"
   ```

   CCP downloads the GeoLite2-City database into HDFS.

Configure Organization Enrichment By Using the Management User Interface

The organization enrichment is an HBase enrichment that provides information about host, user, and any other assets derived from organizational data sources such as CMDB or employee databases. After you establish dataflow to the HBase table, you can use the CCP Management user interface or the CLI to ensure that the enrichment topology is enriching the data flowing past.

Before you begin

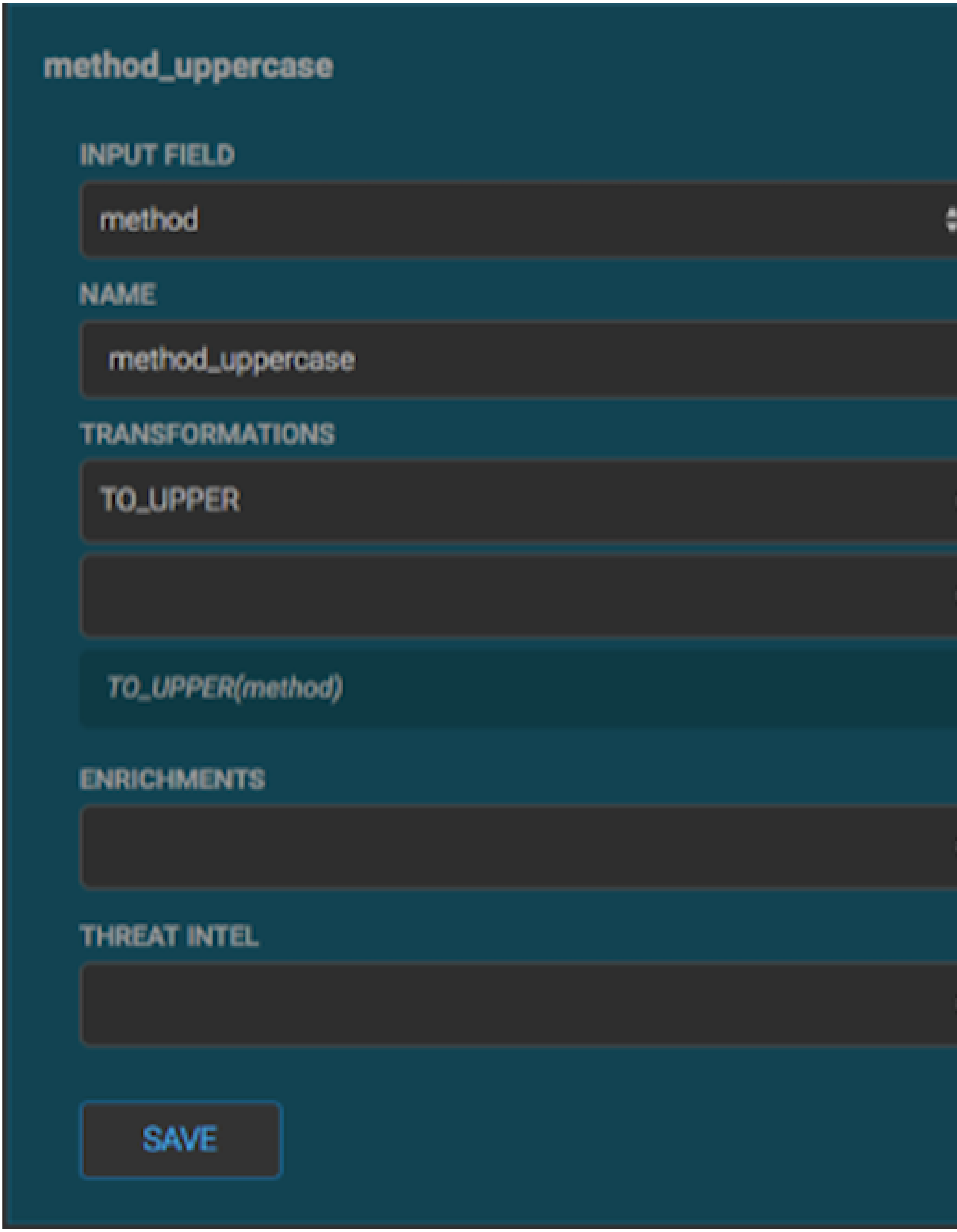
Your sensor must be running and producing data to load sample data.

Procedure

1. From the list of sensors in the Management user interface main window, select your sensor.
2. Click the pencil icon in the toolbar.

   The Management UI displays the sensor panel for the sensor.
3. In the Schema box, click (expand window).

   The Management UI displays a panel for the sensor.



method_uppercase

INPUT FIELD

method

NAME

method_uppercase

TRANSFORMATIONS

TO_UPPER

TO_UPPER(method)

ENRICHMENTS

THREAT INTEL

4. From INPUT FIELD, select the field to transform, enter the name of the new field in the NAME field, and then choose a function with the appropriate parameters in the TRANSFORMATIONS box.
5. From the **ENRICHMENTS** field, choose the appropriate enrichment type.
6. Click **SAVE**.

   If you change your mind and want to remove a transformation, click the "x" next to the field.
7. You can also suppress fields with the transformation feature by clicking ⊘ (suppress icon).

   This icon prevents the field from being displayed, but it does not remove the field entirely.
8. Click **SAVE**.

Configure Organization Enrichment By Using CLI

As an alternative to using the CCP Management user interface to map fields to HBase enrichment, you can use the CLI.

Procedure

1. Edit the new data source enrichment configuration at $METRON_HOME/config/zookeeper/enrichments/$DATASOURCE to associate the ip_src_addr with the user enrichment:

```
{
  "index" : "squid",
  "batchSize" : 1,
  "enrichment" : {
    "fieldMap" : {
      "hbaseEnrichment" : [ "ip_src_addr" ]
    },
    "fieldToTypeMap" : {
      "ip_src_addr" : [ "whois" ]
    },
    "config" : { }
  },
  "threatIntel" : {
    "fieldMap" : { },
    "fieldToTypeMap" : { },
    "config" : { },
    "triageConfig" : {
      "riskLevelRules" : { },
      "aggregator" : "MAX",
      "aggregationConfig" : { }
    }
  },
  "configuration" : { }
}
```

2. Push this configuration to ZooKeeper:

```
$METRON_HOME/bin/zk_load_configs.sh -m PUSH -z $ZOOKEEPER_HOST:2181 -i
 $METRON_HOME/zookeeper
```

What to do next

After you finish enriching telemetry events, you should ensure that the enriched data is displaying on the Metron dashboard.

Batch Loading Enrichment Information

Batch loading is used to load information that does not change frequently. For example, batch loading is ideal for loading from an asset database on a daily or even weekly basis because you don't typically change the number of assets on a network very often.

Enrichment data can be batch loaded from the local file system, HDFS. The enrichment loader transforms the enrichment into a JSON format that is understandable to Metron. The loading framework has additional capabilities for aging data out of the enrichment stores based on time. Once the stores are loaded, you can incorporate an enrichment bolt, that can interact with the enrichment store, into the enrichment topology.

Batch Loading Sources

You can batch load enrichment information from a number of sources.

You can batch load enrichment information from the following sources:

- CSV Flat File Ingestion
- HDFS via MapReduce
- Taxii Loader

CSV File

The shell script $METRON_HOME/bin/flatfile_loader.sh reads data from local disk and loads the enrichment data into an HBase table. This loader uses the special configuration parameter inputFormatHandler to specify how to consider the data. The two implementations are BY_LINE and org.apache.metron.dataloads.extractor.inputformat. WholeFileFormat.

The default is BY_LINE, which makes sense for a list of CSVs in which each line indicates a unit of information to be imported. However, if you are importing a set of STIX documents, then you want each document to be considered as input to the Extractor.

The parameters for the utility are as follows:

Short Code	Long Code	Required?	Description
-h		No	Generates the help screen or set of options
-e	--extractor_config	Yes	JSON document describing the extractor for this input data source
-t	--hbase_table	Yes	The HBase table to import into
-c	--hbase_cf	Yes	The HBase table column family to import into
-i	--input	Yes	The input data location on local disk. If this is a file, then that file is loaded. If this is a directory, then the files are loaded recursively under that directory.
-l	--log4j	No	The log4j properties file to load
-n	--enrichment_config	No	The JSON document describing the enrichments to configure. Unlike other loaders, this is run first if specified.

HDFS Through MapReduce

The shell script $METRON_HOME/bin/flatfile_loader.sh starts the MapReduce job to load data from HDFS to an HBase table. The following is as example of the syntax:

```
$METRON_HOME/bin/flatfile_loader.sh -i /tmp/top-10k.csv -t enrichment -c t -
e ./extractor.json -m MR
```

The parameters for the utility are as follows:

Short Code	Long Code	Is Required?	Description
-h		No	Generate the help screen or set of options
-e	--extractor_config	Yes	JSON document describing the extractor for this input data source
-t	--hbase_table	Yes	The HBase table to import to
-c	--hbase_cf	Yes	The HBase table column family to import to
-i	--input	Yes	The input data location on local disk. If this is a file, then that file is loaded. If this is a directory, then the files are loaded recursively under that directory.
-l	--log4j	No	The log4j properties file to load
-n	--enrichment_config	No	The JSON document describing the enrichments to configure. Unlike other loaders, this is run first if specified.

Taxii Loader

You can use the shell script $METRON_HOME/bin/threatintel_taxii_load.sh to poll a Taxii server for STIX documents and ingest them into HBase.

This Taxii server is often an aggregation server such as Soltra Edge.

This loader requires a configuration file describing the connection information to the Taxii server and Enrichment and Extractor configurations. The following is an example of a configuration file:

```

                        {
   "endpoint" : "http://localhost:8282/taxii-discovery-service"
  ,"type" : "DISCOVER"
  ,"collection" : "guest.Abuse_ch"
  ,"table" : "threat_intel"
  ,"columnFamily" : "cf"
  ,"allowedIndicatorTypes" : [ "domainname:FQDN", "address:IPV_4_ADDR" ]
}
```

endpoint	The URL of the endpoint
type	POLL or DISCOVER, depending on the endpoint
collection	The Taxii collection to ingest.
table	The HBase table to import to.
columnFamily	The column family to import to.

allowedIndicatorTypes	An array of acceptable threat intelligence types

The parameters for the utility are as follows:

Short Code	Long Code	Is Required?	Description
-h		No	Generate the help screen or set of options
-e	--extractor_config	Yes	JSON document describing the extractor for this input data source
-c	--taxii_connection_config	Yes	The JSON configuration file to configure the connection
-p	--time_between_polls	No	The time between polling the Taxii server, in milliseconds. Default: 1 hour
-b	--begin_time	No	Start time to poll the Taxii server (all data from that point will be gathered in the first pull). The format for the date is yyyy-MM-dd HH:mm:ss.
-l	--log4j	No	The Log4j properties to load
-n	--enrichment_config	No	The JSON document describing the enrichments to configure. Unlike other loaders, this is run first if specified.

Configure an Extractor Configuration File

You use the extractor configuration file to batch load the enrichment store into HBase.

Procedure

1. On the host on which Metron is installed, log in as root.
2. Determine the schema of the enrichment source.
3. Create an extractor configuration file called extractor_config_temp.json and populate it with the enrichment source schema:

```
{
 "config" : {
   "columns" : {
      "domain" : 0
      ,"owner" : 1
      ,"home_country" : 2
      ,"registrar": 3
      ,"domain_created_timestamp": 4
   }
   ,"indicator_column" : "domain"
   ,"type" : "whois"
   ,"separator" : ","
  }
  ,"extractor" : "CSV"
}
```

4. Transform and filter the enrichment data as it is loaded into HBase by using Stellar extractor properties in the extractor configuration file.

   CCP supports the following Stellar extractor properties:

value_transform

Transforms fields defined in the columns mapping with Stellar transformations. New keys introduced in the transform are added to the key metadata:

```
"value_transform" : {
   "domain" :
 "DOMAIN_REMOVE_TLD(domain)"
```

value_filter

Allows additional filtering with Stellar predicates based on results from the value transformations. In the following example, records whose domain property is empty after removing the TLD are omitted:

```
"value_filter" : "LENGTH(domain) >
 0",
  "indicator_column" : "domain",
```

indicator_transform

Transforms the indicator column independent of the value transformations. You can refer to the original indicator value by using indicator as the variable name, as shown in the following example:

```
"indicator_transform" : {
   "indicator" :
 "DOMAIN_REMOVE_TLD(indicator)"
```

In addition, if you prefer to piggyback your transformations, you can refer to the variable domain, which allows your indicator transforms to inherit transformations to this value.

indicator_filter

Allows additional filtering with Stellar predicates based on results from the value transformations. In the following example, records with empty indicator values after removing the TLD are omitted:

```
"indicator_filter" :
 "LENGTH(indicator) > 0",
  "type" : "top_domains",
```

Including all of the supported Stellar extractor properties in the extractor configuration file, looks similar to the following:

```
{
   "config" : {
     "zk_quorum" : "$ZOOKEEPER_HOST:2181",
     "columns" : {
        "rank" : 0,
        "domain" : 1
     },
     "value_transform" : {
        "domain" : "DOMAIN_REMOVE_TLD(domain)"
     },
     "value_filter" : "LENGTH(domain) > 0",
     "indicator_column" : "domain",
     "indicator_transform" : {
        "indicator" : "DOMAIN_REMOVE_TLD(indicator)"
     },
     "indicator_filter" : "LENGTH(indicator) > 0",
     "type" : "top_domains",
     "separator" : ","
   },
   "extractor" : "CSV"
 }
```

If you run a file import with this data and extractor configuration, you get the following two extracted data records:

Indicator	Type	Value
google	top_domains	{ "rank" : "1", "domain" : "google" }
yahoo	top_domains	{ "rank" : "2", "domain" : "yahoo" }

5. To access properties that reside in the global configuration file, provide a ZooKeeper quorum by using the zk_quorum property.

   If the global configuration looks like "global_property" : "metron-ftw", enter the following to expand the value_transform:

```
"value_transform" : {
    "domain" : "DOMAIN_REMOVE_TLD(domain)",
     "a-new-prop" : "global_property"
 },
```

   The resulting value data looks like the following:

Indicator	Type	Value
google	top_domains	{ "rank" : "1", "domain" : "google", "a-new-prop" : "metron-ftw" }
yahoo	top_domains	{ "rank" : "2", "domain" : "yahoo", "a-new-prop" : "metron-ftw" }

6. Remove any non-ASCII invisible characters included when you cut and pasted the value_transform information:

```
iconv -c -f utf-8 -t ascii extractor_config_temp.json -o
 extractor_config.json
```

   The extractor_config.json file is not stored by the loader. If you want to reuse it, you must store it yourself.

Configure Element-to-Enrichment Mapping

Configure which element of a tuple should be enriched with which enrichment type. This configuration is stored in Apache ZooKeeper.

Procedure

1. On the host with Metron installed, log in as root.
2. Cut and paste the following syntax into a file called enrichment_config_temp.json, being sure to customize $ZOOKEEPER_HOST and $DATASOURCE to your specific values, where $DATASOURCE refers to the name of the data source you use to bulk load the enrichment:

```
{
     "zkQuorum" : "$ZOOKEEPER_HOST:2181"
    ,"sensorToFieldList" : {
          "$DATASOURCE" : {
             "type" : "ENRICHMENT"
            ,"fieldToEnrichmentTypes" : {
                 "domain_without_subdomains" : [ "whois" ]
              }
          }
    }
}
```

3. Remove any non-ASCII invisible characters in the pasted syntax in Step 2:

```
iconv -c -f utf-8 -t ascii enrichment_config_temp.json -o
 enrichment_config.json
```

Run the Enrichment Loader

After you configure the extractor configuration file and the element-enrichment mapping, you must run the loader to move the data from the enrichment source to the CCP enrichment store and store the enrichment configuration in Apache ZooKeeper.

Procedure

1. If you will be bulk loading enrichments using the MR mode, add the metron user to the access control list of users and groups in the Yarn Queue Manager.

   You do not need to perform this step if you use the LOCAL mode.

   a) In Ambari, click (Ambari views) and choose **Yarn Queue Manager**.

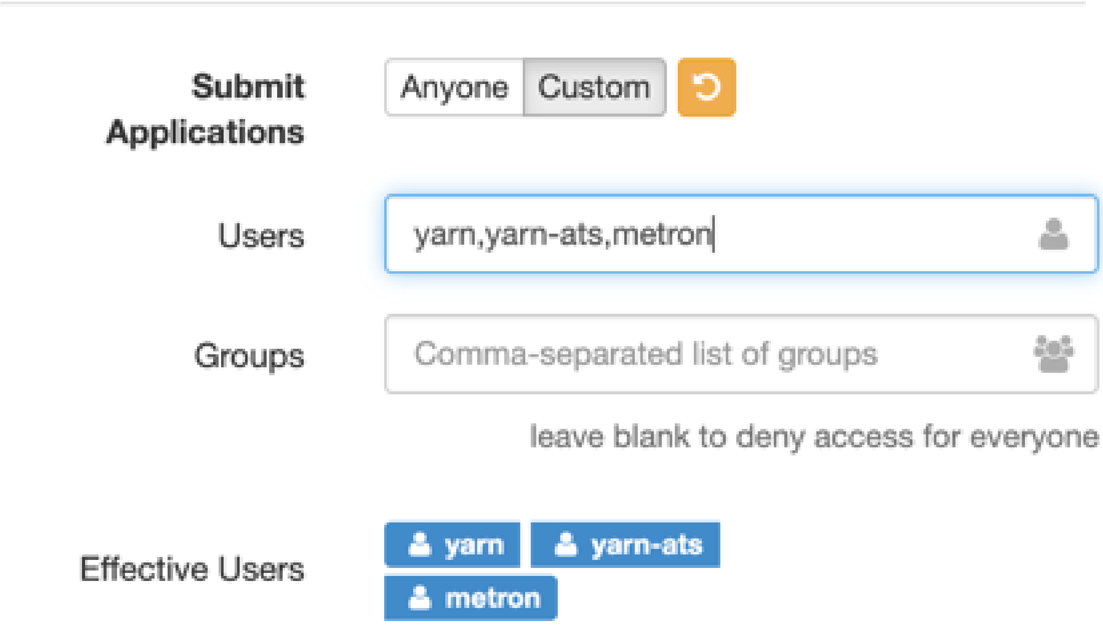
   b) In the upper left corner of the window, under **Add Queue**, select **default**.

      Ambari displays the default information on the right side of the window.

   c) By **Submit Applications**, click **Custom**.

   d) In the **Users** field, add metron.

      For example:



Submit Applications: Custom

Users: yarn,yarn-ats,metron

Groups: Comma-separated list of groups

leave blank to deny access for everyone

Effective Users: yarn yarn-ats metron

2. Use the loader to move the enrichment source to the enrichment store in ZooKeeper:

   ```
   $METRON_HOME/bin/flatfile_loader.sh -n enrichment_config.json -i
    whois_ref.csv -t enrichment -c t -e extractor_config.json
   ```

   CCP loads the enrichment data into Apache HBase and establishes a ZooKeeper mapping. The data is extracted using the extractor and the configuration is defined in the extractor_config.json file and populated into an HBase table called enrichment.
3. Verify that the logs were properly ingested to HBase:

   ```
   hbase shell
   scan 'enrichment'
   ```
4. Verify that the ZooKeeper enrichment tag was properly populated:

   ```
   $METRON_HOME/bin/zk_load_configs.sh -m DUMP -z $ZOOKEEPER_HOST:2181
   ```
5. Generate some data by using a client for your particular data source to execute requests.

Stream Enrichment Information

Streaming enrichment information is useful when you need enrichment information in real time. This type of information is most useful in real time as opposed to waiting for a bulk load of the enrichment information. You incorporate streaming intelligence feeds slightly differently than when you use bulk loading. The enrichment information resides in its own parser topology instead of in an extraction configuration file. The parser file defines the input structure and how that data is used in enrichment. Streaming information goes to Apache HBase rather than to Apache Kafka, so you must configure the writer by using both the writerClassName and simple HBase enrichment writer (shew) parameters.

Procedure

1. Define a parser topology in $METRON_HOME/zookeeper/parsers/user.json:

   ```
   touch $METRON_HOME/config/zookeeper/parsers/user.json
   ```
2. Populate the file with the parser topology definition.

   For example, the following commands associate IP addresses with user names for the Squid information.

   ```
   {
    "parserClassName" : "org.apache.metron.parsers.csv.CSVParser"
    ,"writerClassName" :
    "org.apache.metron.enrichment.writer.SimpleHbaseEnrichmentWriter"
    ,"sensorTopic":"user"
    ,"parserConfig":
    {
       "shew.table" : "enrichment"
      ,"shew.cf" : "t"
      ,"shew.keyColumns" : "ip"
      ,"shew.enrichmentType" : "user"
      ,"columns" : {
         "user" : 0
        ,"ip" : 1
                   }
    }
   }
   ```

parserClassName

The parser name.

writerClassName

The writer destination. For streaming parsers, the destination is SimpleHbaseEnrichmentWriter.

sensorTopic

Name of the sensor topic.

shew.table

The simple HBase enrichment writer (shew) table to which you want to write.

shew.cf

The simple HBase enrichment writer (shew) column family.

shew.keyColumns

The simple HBase enrichment writer (shew) key.

shew.enrichmentType

The simple HBase enrichment writer (shew) enrichment type.

columns

The CSV parser information. In this example, the user name and IP address.

This file fully defines the input structure and how that data can be used in enrichment.

3. Push the configuration file to Apache ZooKeeper:

a) Create a Kafka topic sized to manage your estimated data flow:

```
/usr/hdp/current/kafka-broker/bin/kafka-topics.sh --create --zookeeper
 $ZOOKEEPER_HOST:2181 --replication-factor 1 --partitions 1 --topic user
```

Push the configuration file to ZooKeeper:

```
$METRON_HOME/bin/zk_load_configs.sh -m PUSH -z $ZOOKEEPER_HOST:2181 -i
 $METRON_HOME/zookeeper
```

4. Start the user parser topology:

```
$METRON_HOME/bin/start_parser_topology.sh -s user -z $ZOOKEEPER_HOST:2181
 -k $KAKFA_HOST:6667
```

The parser topology listens for data streaming in and pushes the data to HBase. Data is flowing into the HBase table, but you must ensure that the enrichment topology can be used to enrich the data flowing past.

5. Edit the new data source enrichment configuration at $METRON_HOME/config/zookeeper/enrichments/squid to associate the ip_src_addr with the user name:

```
{
  "enrichment" : {
    "fieldMap" : {
      "hbaseEnrichment" : [ "ip_src_addr" ]
    },
    "fieldToTypeMap" : {
      "ip_src_addr" : [ "user" ]
    },
    "config" : { }
  },
  "threatIntel" : {
    "fieldMap" : { },
    "fieldToTypeMap" : { },
    "config" : { },
    "triageConfig" : {
      "riskLevelRules" : { },
      "aggregator" : "MAX",
      "aggregationConfig" : { }
    }
  },
  "configuration" : { }
}
```

6. Push the new data source enrichment configuration to ZooKeeper:

```
$METRON_HOME/bin/zk_load_configs.sh -m PUSH -z $ZOOKEEPER_HOST:2181 -i
 $METRON_HOME/zookeeper
```