Understanding Parsing

Date of Publish: 2019-04-09

https://docs.hortonworks.com

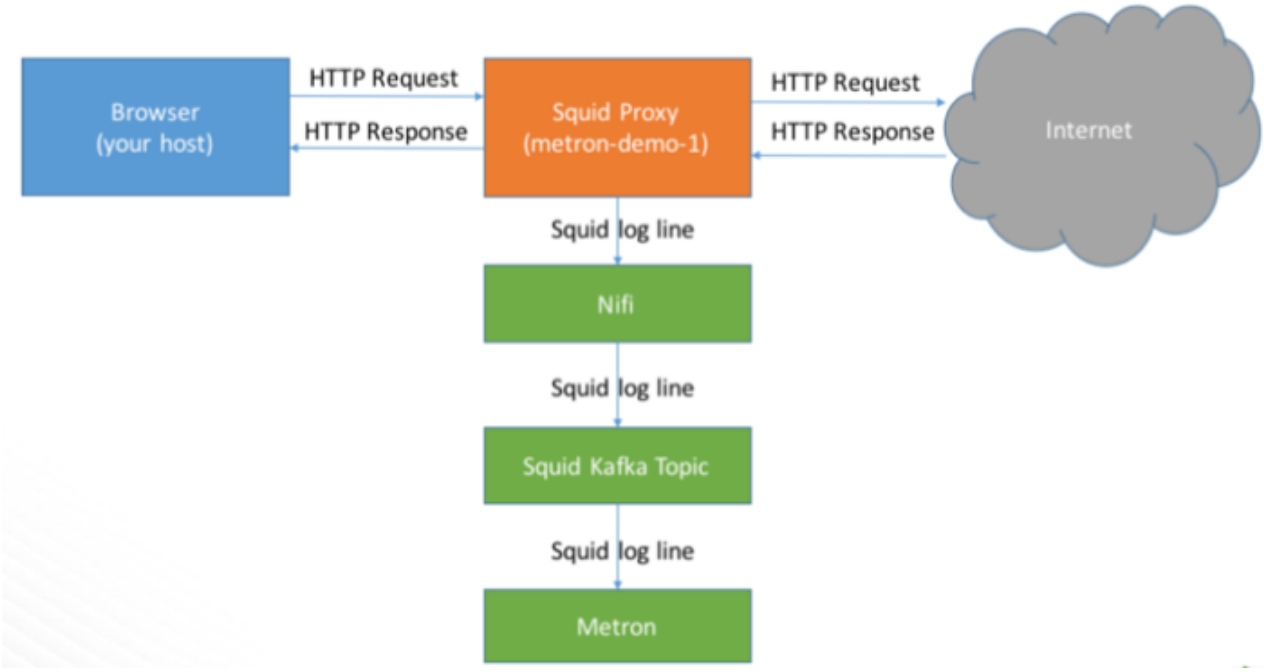
Contents

Understanding Parsers

Parsers are pluggable components that transform raw data (textual or raw bytes) into JSON messages suitable for downstream enrichment and indexing.

Data flows through the parser bolt via Apache Kafka and into the enrichments topology in Apache Storm. Errors are collected with the context of the error (for example, stacktrace) and the original message causing the error and are sent to an error queue. Invalid messages as determined by global validation functions are also treated as errors and sent to an error queue.

For example, for a Squid parser, NiFi ingests the contents of the Squid proxy access log, the parser transforms the contents of the log, converts it to json, and inserts it into a Squid Kafka topic, which is then passed on to Metron.



HCP supports two types of parsers: general purpose and Java.

Java Parsers

The Java parser is written in Java and conforms with the MessageParser interface. This kind of parser is optimized for speed and performance and is built for use with higher-velocity topologies.

Java parsers are not easily modifiable; to make changes to them, you must recompile the entire topology.

Currently, the Java adapters included with HCP are as follows:

- org.apache.metron.parsers.ise.BasicIseParser
- org.apache.metron.parsers.bro.BasicBroParser
- org.apache.metron.parsers.sourcefire.BasicSourcefireParser
- org.apache.metron.parsers.lancope.BasicLancopeParser

General Purpose Parsers

The general-purpose parser is primarily designed for lower-velocity topologies or for quickly setting up a temporary parser for a new telemetry.

General purpose parsers are defined using a config file, and you need not recompile the topology to change them. HCP supports two general purpose parsers: Grok and CSV.

Grok parser

The Grok parser class name (parserClassName) is org.apache.metron,parsers.GrokParser.

Grok has the following entries and predefined patterns for parserConfig:

grokPath	The path in HDFS (or in the Jar) to the grok statement. By default attempts to load from HDFS, then falls back to the classpath, and finally throws an exception if unable to load a pattern.
patternLabel	The pattern label to use from the Grok statement.
timestampField	The field to use for timestamp. If your data does not have a field exactly named "timestamp" this field is required, otherwise the record will not pass validation. If the timestampField is included in the list of timeFields, it will first be parsed using the provided dateFormat.
timeFields	A list of fields to be treated as time.
dateFormat	The date format to use to parse the time fields. Default is "yyyy-MM-dd HH:mm:ss.S z".
timezone	The timezone to use. UTC is the default.

CSV Parser

The CSV parser class name (parserClassName) is org.apache.metron.parsers.csv.CSVParser

CSV has the following entries and predefined patterns for parserConfig:

timestampFormat	The date format of the timestamp to use. If unspecified, the parser assumes the timestamp is starts at UNIX epoch.
columns	A map of column names you wish to extract from the CSV to their offsets. For example, { 'name' : 1,'profession' : 3} would be a column map for extracting the 2nd and 4th columns from a CSV.
separator	The column separator. The default value is ",".

JSON Map Parser

The JSON parser class name (parserClassName) is org.apache.metron.parsers.csv.JSONMapParser

JSON has the following entries and predefined patterns for parserConfig:

mapStrategy	A strategy to indicate how to handle multi-dimensional Maps. This is one of:
	DROP — Drop fields which contain maps
	UNFOLD — Unfold inner maps. So { "foo" : { "bar" :

	1} } would turn into {"foo.bar" : 1}
ALLOW	Allow multidimensional maps
ERROR	Throw an error when a multidimensional map is encountered

timestamp	This field is expected to exist and, if it does not, then current time is inserted.
jsonQuery	If this JSON query string is present, the result of the query will be a list of messages. This is useful if you have a JSON document that contains a list or array of messages embedded in it, and you do not have another means of splitting the message.

Parser Configuration

The configuration for the various parser topologies is defined by JSON documents stored in ZooKeeper.

The JSON document consists of the following attributes:

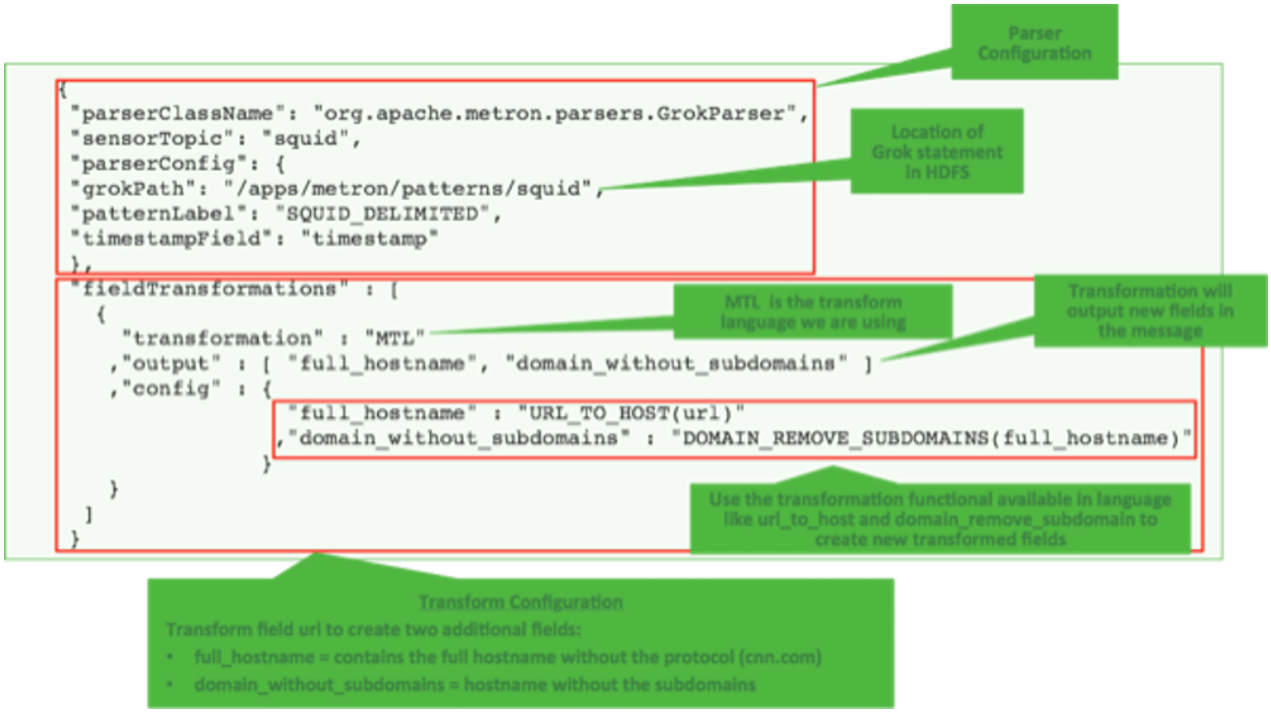
parserClassName	The fully qualified class name for the parser to be used.
sensorTopic	The Kafka topic to send the parsed messages to.
parserConfig	A JSON Map representing the parser implementation specific configuration.
fieldTransformations	An array of complex objects representing the transformations to be done on the message generated from the parser before writing out to the Kafka topic. The fieldTransformations is a complex object which defines a transformation that can be done to a message. This transformation can perform the following: • Modify existing fields to a message • Add new fields given the values of existing fields of a message • Remove existing fields of a message

Example: fieldTransformation Configuration

The fieldTransformation is a complex object which defines a transformation that can be done to a message.

In this example, the host name is extracted from the URL by way of the URL_TO_HOST function. Domain names are removed by using DOMAIN_REMOVE_SUBDOMAINS, thereby creating two new fields (full_hostname and domain_without_subdomains) and adding them to each message.

Configuration File with Transformation Information



The format of a fieldTransformation is as follows:

input

An array of fields or a single field representing the input. This is optional; if unspecified, then the whole message is passed as input.

output

The outputs to produce from the transformation. If unspecified, it is assumed to be the same as inputs.

transformation

The fully qualified class name of the transformation to be used. This is either a class which implements FieldTransformation or a member of the FieldTransformations enum.

config

A String to Object map of transformation specific configuration.

HCP currently implements the following fieldTransformations options:

REMOVE

This transformation removes the specified input fields. If you want a conditional removal, you can pass a Metron Query Language statement to define the conditions under which you want to remove the fields.

The following example removes field1 unconditionally:

```
{
...
    "fieldTransformations" : [
          {
            "input" : "field1"
          , "transformation" :
 "REMOVE"
          }
                      ]
```

```
}
```

The following example removes field1 whenever field2 exists and has a corresponding value equal to 'foo':

```
{
...
  "fieldTransformations" : [
          {
            "input" : "field1"
          , "transformation" :
 "REMOVE"
          , "config" : {
              "condition" :
 "exists(field2) and field2 ==
 'foo'"
                        }
          }
                        ]
}
```

IP_PROTOCOL

This transformation maps IANA protocol numbers to consistent string representations.

The following example maps the protocol field to a textual representation of the protocol:

```
{
...
    "fieldTransformations" : [
          {
            "input" : "protocol"
          , "transformation" :
 "IP_PROTOCOL"
          }
                        ]
}
```

STELLAR

lo

This transformation executes a set of transformations expressed as Stellar Language statements.

The following example adds three new fields to a message:

utc_timestamp	The UNIX epoch timestamp based on the timestamp field, a dc field which is the data center the message comes from and a dc2tz map mapping data centers to timezones.
url_host	The host associated with the url in the url field.
url_protocol	The protocol associated with the url in the url field.

```
{
```

```
...
    "fieldTransformations" : [
          {
           "transformation" :
 "STELLAR"
          ,"output" :
 [ "utc_timestamp", "url_host",
 "url_protocol" ]
          ,"config" : {
            "utc_timestamp" :
 "TO_EPOCH_TIMESTAMP(timestamp,
 'yyyy-MM-dd
HH:mm:ss', MAP_GET(dc, dc2tz,
 'UTC') )"
          ,"url_host" :
 "URL_TO_HOST(url)"
          ,"url_protocol" :
 "URL_TO_PROTOCOL(url)"
                      }
          }
                      ]
   ,"parserConfig" : {
      "dc2tz" : {
                 "nyc" : "EST"
                ,"la" : "PST"
                ,"london" : "UTC"
                 }
    }
}
```

Note that the dc2tz map is in the parser config, so it is accessible in the functions.