Notes On Clouder morphline parsing

I found that adapting and quickly on-boarding myself to the Morphlines framework very frustrating. The purpose of this document is to serve as supplement material to additional available resources.

The platform I work on uses Apache Nifi for our data ingestion, and we use our parsers to filter the incoming data received by the pipeline. Apache NIFI provides a nice UI to help us visually see what data topics are on-boarded and are currently being ingested.

AVRO Files (.avsc): The avro schema defines the column names (aka field names), default data type (String, Integer, Long, Null) and default value in the Hadoop. It stores the data definition (schema) in a JSON format. After we store the table in Hadoop, we will create a table in HIVE or Impala. This data can then be queried using standard SQL. Notably, the data scientists, ML engineers, and data analytics can then proceed with reading and analyzing the incoming data. The format for this is JSON.

A sample avro file could be like this:

{

"type" : "record",

"name" : "AutoGeneratedSchema",

"doc" : "Sqoop import of QueryResult",

"fields" : [ {

"name" : "ID",

"type" : [ "null", "string" ],

"default" : null,

"columnName" : "ID",

"sqlType" : "2"

}, {

"name" : "NAME",

"type" : [ "null", "string" ],

"default" : null,

"columnName" : "NAME",

"sqlType" : "12"

}, {

"name" : "AGE",

"type" : [ "null", "string" ],

"default" : null,

"columnName" : "AGE",

"sqlType" : "2"

}, {

"name" : "GEN",

"type" : [ "null", "string" ],

"default" : null,

"columnName" : "GEN",

"sqlType" : "12"

}, {

"name" : "CREATE\_DATE",

"type" : [ "null", "long" ],

"default" : null,

"columnName" : "CREATE\_DATE",

"sqlType" : "93"

}, {

"name" : "PROCESS\_NAME",

"type" : [ "null", "string" ],

"default" : null,

"columnName" : "PROCESS\_NAME",

"sqlType" : "12"

}, {

"name" : "UPDATE\_DATE",

"type" : [ "null", "long" ],

"default" : null,

"columnName" : "UPDATE\_DATE",

"sqlType" : "93"

} ],

"tableName" : "QueryResult"

}

Configuration files (.conf). This file is responsible for the parsing of the data files that are coming in. In my experience I work with these types of incoming data files:

CEF

Delimitted

Key Value Pair

JSON

In some instances you may work with a hybrid of these data formats such as JSON with nested delimited. An example would be this:

{

“Feed\_Name” : “<25> 2020-09-03T10:51:32:21233:2423 feed\_source feed\_attribute 5231321 ‘\SQLGUARD ALERT’|, \field\_name\_one=1234567|field\_name\_two=1234567|field\_name\_three=1234567|instance\_name=ThisiTheEnd|msg=\”,

“hostname” : “mockHostname”,

“environment” : “production”,

“timestamp” : “12312321321”,

“data\_source” : “foreign”,

“mockFieldName” : “mock”,

}

{

“Feed\_Name” : “<25> 2020-09-03T10:51:32:21233:2423 feed\_source feed\_attribute 5231321 ‘\SQLGUARD ALERT’|, \field\_name\_one=1234567|field\_name\_two=1234567|field\_name\_three=1234567|instance\_name=ThisiTheEnd|msg=\”,

“hostname” : “mockHostname”,

“environment” : “production”,

“timestamp” : “12312321321”,

“data\_source” : “foreign”,

“mockFieldName” : “mock”,

}

{

“Feed\_Name” : “<25> 2020-09-03T10:51:32:21233:2423 feed\_source feed\_attribute 5231321 ‘\SQLGUARD ALERT’|, \field\_name\_one=1234567|field\_name\_two=1234567|field\_name\_three=1234567|instance\_name=ThisiTheEnd|msg=\”,

“hostname” : “mockHostname”,

“environment” : “production”,

“timestamp” : “12312321321”,

“data\_source” : “foreign”,

“mockFieldName” : “mock”,

}

In this example, we are only interested in the JSON filed name: <random> and want to extract the values where the key is “Feed\_Name”. This would be a combination of JSON with pipe delimited. (I think in an ideal world, all the incoming data would be a standard format (JSON, XML, CSV) however this is not always the case.

A breakdown of the conf file.

First glimpse of this file, it sort of reminds me of Java and C++. Please not the following

-This type of file has sequential execution. The order matters. Commands and instructions are executed in a top – down approach.

-A few things to note: variable assignment in a conf is not the same as a standard programming language. You cannot assign a variable X, and another variable Y. Apply changes to X and then re-assign that value to variable Y. Rather all changes are applied to incoming input source.

-The main purpose of variables is for printing to the output console.

-You can provide sections where you utilize Java. This is very useful if you are working with a very unique type of data format and need to handle special cases. (I think in an ideal world there would be universal formatting and consistency for input data files).

Reference Documentation:

<https://blog.cloudera.com/introducing-morphlines-the-easy-way-to-build-and-integrate-etl-apps-for-hadoop/>

<http://kitesdk.org/docs/1.1.0/morphlines/morphlines-reference-guide.html>

<https://grokdebug.herokuapp.com> :: Use this if you are trying to parse through a string. Grokking is format, case and space sensitive. It is good practice to remove confidential data before placing your data into the online editor. Note: GREEDYDATA consumes everything

<https://regex101.com>