University of East London
School of Architecture, Computing and Engineering (ACE)
Author: Dr Amin Karami (a.karami@uel.ac.uk) 25-26, Big Data Analytics

Tutorial 5: Working with Unstructured Data in Hadoop

CN7031 - Big Data Analytics

Dr Amin Karami (a.karami@uel.ac.uk)

LEARNING OUTCOMES: After completing this tutorial, you should:

- Gain the ability to work with unstructured data in Hadoop.
- Develop the skills to create tables in Hive, allowing for structured querying of unstructured data.
- Acquire advanced techniques and tricks in Hive to optimize productivity when working with unstructured data.
- Learn how to leverage Impala for fast and interactive SQL queries on unstructured data.

Phase 1: Bulk Upload Data

Step 1: Copy data to HDFS

Download data from here (OneDrive login is needed) and copy it in the HDFS. This is an access log data including 5m records, with the size of ~1GB.

```
$ hdfs dfs -mkdir -p /user/hive/warehouse/access_log

$ hdfs dfs -put -f /home/cloudera/Desktop/access.log
/user/hive/warehouse/access_log

$ hdfs dfs -chmod -R 777 /user/hive/warehouse/access_log
```

Step 2: Verify data is in HDFS and Delete original data from disk

```
$ hdfs dfs -ls /user/hive/warehouse/access_log

$ rm -f /home/cloudera/Desktop/access.log
```

Step 3: Create a Table in Hive

You can create a table in Hive and then use Impala and Hue to query the data. The table creation process involves two steps. First, you'll use **Hive's SerDes (Serializer/Deserializer)** to parse the logs into separate fields using a regular expression. Then, you'll transfer the data from the intermediate table to a final table that doesn't require any special SerDe. Once the data is in the final table, you can query it faster and more interactively using Impala.

What is Hive's SerDes? It is a critical component in Hive that acts as an interface between the binary data stored in HDFS and the metadata of Hive tables. It allows data to be interpreted as tabular data, enabling Hive to read and write data in various formats such as CSV, JSON, Avro, ORC, and Parquet. The SerDe library in Hive provides both built-in and custom SerDes, allowing users to define their own SerDes for handling data in custom formats. The SerDe instructs Hive on how to process each record (row) by specifying the serialization and deserialization methods. This allows you to structure the data in a way that can be efficiently queried. Once the data is in the table, you can use Impala and Hue to query it faster and more interactively.

Go to the Hive Editor and do the followings back-to-back:

a) Drop an existing table:

```
DROP TABLE IF EXISTS intermediate_logs;
```

b) Create an intermediate table:

```
CREATE EXTERNAL TABLE intermediate_logs (
    ip STRING,
    datee STRING,
    method STRING,
    url STRING,
    http_version STRING,
    code1 STRING,
    code2 STRING,
    dash STRING,
    user_agent STRING)
ROW FORMAT SERDE 'org.apache.hadoop.hive.contrib.serde2.RegexSerDe'
WITH SERDEPROPERTIES (
    'input.regex' = '([^ ]*) - - \\[(\\d{2}/[A-Za-z]{3}/\\d{4}:\\d{2}:\\d{2}:\\d{2} -\\d{4})\\] "([^\ ]*) ([^\ ]*) ([^\ ]*)" (\\d*) (\\d*) "([^"]*)" "([^"]*)"',
    'output.format.string' = "%1$$s %2$$s %3$$s %4$$s %5$$s %6$$s %7$$s %8$$s %9$$s")
LOCATION '/user/hive/warehouse/access log';
```

Code format:

```
CREATE EXTERNAL TABLE intermediate_logs (
    ip STRING,
    datee STRING,
    method STRING,
    url STRING,
    http_version STRING,
    code1 STRING,
    code2 STRING,
    dash STRING,
    user_agent STRING)
ROW FORMAT SERDE 'org.apache.hadoop.hive.contrib.serde2.RegexSerDe'
WITH SERDEPROPERTIES (
    'input.regex' = '([^ ]*) - - \\[(\\d{2}/[A-Za-z]{3}/\\d{4}:\\d{2}:\\d{2}:\\d{2} -\\d{4})\\] "([^\ ]*) ([^\ ]*) ([^\ ]*)" (\\d*) (\\d*) "([^"]*)"
"([^"]*)"',
    'output.format.string' = "%1$$s %2$$s %3$$s %4$$s %5$$s %6$$s %7$$s %8$$s %9$$s")
LOCATION '/user/hive/warehouse/access_log';
```

Sample of log:

48.138.27.161 - - [31/Mar/2015:21:53:15 -0400] "GET /home HTTP/1.1" 200 570 "-" "Mozilla/5.0 (Windows NT 6.1; WOW64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/35.0.1916.153 Safari/537.36"

What is **'input.regex'?** This property defines the regular expression pattern used to parse the input data and extract the columns. Let's break down the regular expression pattern:

- `([^ ]*)`: This part matches any sequence of characters that does not contain a space character. The parentheses () capture the matched sequence as a group, which corresponds to the first column in the table (ip).
- `- -`: This part matches the literal string " - - "
- `\\[(\\d{2}/[A-Za-z]{3}/\\d{4}:\\d{2}:\\d{2}:\\d{2} -\\d{4})\\]`: This part matches a sequence of characters enclosed in square brackets. The parentheses () capture the matched sequence as a group, which corresponds to the second column in the table (datee). The backslashes `\\` are used to escape the square brackets, as they have special meaning in regular expressions.
- `"([^\ ]*)`: This part matches any sequence of characters that does not contain a space character, enclosed within double quotes. The parentheses () capture the matched sequence as a group, which corresponds to the third column in the table (method).
- `([^\ ]*)`: This part matches any sequence of characters that does not contain a space character. The parentheses () capture the matched sequence as a group, which corresponds to the fourth column in the table (url).
- `([^\ ]*)"`: This part matches any sequence of characters that does not contain a space character. The parentheses () capture the matched sequence as a group, which corresponds to the fifth column in the table (http_version).
- `(\\d*)`: This part matches any sequence of digits. The parentheses () capture the matched sequence as a group, which corresponds to the sixth column in the table (code1 and code2).
- `"([^"]*)"`: This part matches any sequence of characters enclosed within double quotes. The parentheses () capture the matched sequence as a group, which corresponds to the eighth column in the table (dash).
- `"([^"]*)"`: This part matches any sequence of characters enclosed within double quotes. The parentheses () capture the matched sequence as a group, which corresponds to the ninth column in the table (user_agent).

c) Drop an existing table:

```
DROP TABLE IF EXISTS logs;
```

d) Create the logs table:

```
CREATE EXTERNAL TABLE logs (
    ip STRING,
    datee STRING,
    method STRING,
    url STRING,
    http_version STRING,
    code1 STRING,
    code2 STRING,
    dash STRING,
    user_agent STRING)
ROW FORMAT DELIMITED FIELDS TERMINATED BY ','
LOCATION '/user/hive/warehouse/logs';

ADD JAR /usr/lib/hive/lib/hive-contrib.jar;

INSERT OVERWRITE TABLE logs SELECT * FROM intermediate_logs;
```

The hive-contrib.jar is a JAR file that contains additional contributed functionality for Apache Hive. It includes various extensions, utilities, and additional features that are not part of the core Hive distribution. These contributed features are developed and maintained by the Hive community. It includes additional UDFs (User-Defined Functions), SerDes (Serializer/Deserializer), input/output formats, and other utilities that can enhance the functionality and capabilities of Hive.

Step 4: Reload metadata in Impala:

To inform Impala about tables created with a different tool, you need to refresh the metadata information. This can be done using a command that forces Impala to reload metadata from the underlying storage system. Doing so ensures that any changes made to the schema or data are reflected in Impala's metadata cache.

Go to the Impala Editor and do the following:

```
invalidate metadata;
```

Now, data is ready for analysis using SQL queries.

Step 5: Data Analysis using SQL in Impala:

- Task 1. Find the top 10 most accessed URLs:

```
SELECT url, COUNT(*) AS access_count
FROM logs
GROUP BY url
ORDER BY access_count DESC
LIMIT 10;
```

- Task 2. Identify the most common user agents:

```
SELECT user_agent, COUNT(*) AS count
FROM logs
GROUP BY user_agent
ORDER BY count DESC
LIMIT 5;
```

- Task 3. Determine the busiest hour of the day:

```
SELECT SUBSTRING(datee, 13, 2) AS hour, COUNT(*) AS request_count
FROM logs
GROUP BY hour
ORDER BY request_count DESC
LIMIT 1;
```

- Task 4. Find IP addresses that have made more requests to more than 35 distinct URLs:

```
SELECT ip, COUNT(DISTINCT url) AS distinct_url_count
FROM logs
GROUP BY ip
HAVING distinct_url_count > 35
ORDER BY distinct_url_count DESC;
```