HCatalog

Data and Analytics Training

Big Data – Self-Study Guide (208)

This study guide serves as a basis for your self-study on the basics of HCatalog. The study guide gives you an overview on the main topics and presents a number of internal and external resources you should use to get a deeper understanding of HCatalog, the related terms and technical details.

Please especially use the external resources provided. They are a main part of this study guide and are required to get a good understanding of the discussed topics.

Estimated Completion Time

2 hour (including time for external resources)

Objectives

After completing this study guide, you will be able to understand

- Making your scripts independent of HDFS file locations
- Some examples of how HCatalog makes working with different technologies easier

Contents

Introduction

You've gone through quite a bit so far processing so many Map/Reduce jobs that you've been doing, you started to get some kind of querying like functionality in the module on Hive. In this module you are going to take a look at other cool big data term, HCatalog. You are actually going to look at how to minimise hard-coded dependencies in the processing of big data jobs.

Let's talk a little bit about what's involved generally with big data processing. Of course, it varies depending on what it is you are actually trying to achieve. Every business has its own different ways of doing things, but generally the approach is, you take some source data, upload it to HDFS, then run some sort of Map/Reduce process on that. That might be one Map/Reduce job or that might be a sequence of things you do. And it might be a combination of custom Map/Reduce code. It might be Hive, it might be Pig. There's a whole bunch of thing that you could do to get that data into the shape you want it, and then typically consume that data into some sort of analysis to get insight, or some set of results.

That's kind of the pattern in big data processing. And the other thing to emphasise is that HDInsight is a cloud service. It is provided on a pay-as-you-use basis in the Windows Azure cloud stack. The idea is that, you might want to be able to use the HDInsight cluster to do processing when you need it, and then just take the cluster down when you don't need it, or perhaps even more complex scenario might be even fire up the bigger HDInsight cluster with more nodes, to do a particularly big job and then take it down and fire back up again for the smaller number of nodes just for some post-processing. In this way you are reducing the cost. And not paying for the compute nodes that you are not actually using. And most companies these days appreciate the idea they are only going to pay for what they use, nothing more. But this can break some scripts that have hard-coded HDFS paths. Let's see how HCatalog can support to make things work by not relying on hard-coded paths on the HDFS.

Understanding HCatalog

Apache HCatalog is a table and storage management layer for Hadoop that enables users with different data processing tools – Apache Pig, Apache MapReduce, and Apache Hive – to more easily read and write data on the grid. HCatalog's table abstraction presents users with a relational view of data in the Hadoop Distributed File System (HDFS) and ensures that users need not worry about where or in what format their data is stored. HCatalog displays data from RCFile format, text files, or sequence files in a tabular view. It also provides REST APIs so that external systems can access these tables' metadata.

HCatalog supports reading and writing files in any format for which a Hive SerDe (serializer-deserializer) can be written. By default, HCatalog supports RCFile, CSV, JSON, and SequenceFile formats. To use a custom format, you must provide the InputFormat, OutputFormat, and SerDe.

HCatalog is built on top of the Hive metastore and incorporates components from the Hive DDL. HCatalog provides read and write interfaces for Pig and MapReduce and uses Hive's command line interface for issuing data definition and metadata exploration commands. It also presents a REST interface to allow external tools access to Hive DDL (Data Definition Language) operations, such as "create table" and "describe table".

HCatalog presents a relational view of data. Data is stored in tables and these tables can be placed into databases. Tables can also be partitioned on one or more keys. For a given value of a key (or set of keys) there will be one partition that contains all rows with that value (or set of values).[1]

```
SourceData = LOAD '/data/source' USING PigStorage(',') AS (col1,
col2:float);


SortedData = ORDER SourceData BY col1 ASC;
STORE SortedData INTO '/data/output';
```

For example, if you look at the Pig script above. In this case, you've got the source data (a relation), you are loading that from the /data/source folder, you are saying that's a comma delimited file and you are specifying it contains one column that is a character array, and one column that is a float. And then you are sorting it by column1 and then you are storing the output into another folder /data/output.

Now the interesting thing here is, the only thing that Pig needs to know is that the data has a column called col1. It doesn't need to be called col1. It is just the column to order the data by. What you could actually do is, you could abstract that storage and abstract the schema by creating a Hive table.

```
SourceData = LOAD 'StagingTable' USING
org.apache.hcatalog.pig.HCatLoader();

SortedData = ORDER SourceData BY col1 ASC;

STORE SortedData INTO 'OutputTable' USING
org.apache.hcatalog.pig.HCatStorer();
```

[1] http://hortonworks.com/hadoop/hcatalog/

If you create a Hive table called StagingTable, and you look at the revised code above where you are using HCatalog, the SourceData now just loads the staging table. This Pig script, doesn't need to know where the underlying table is, that's part of the definition of the StagingTable Hive table. And the Pig script simply loads data out of the Hive table. And you are using the hcatalog.pig.HCatLoader rather than that PigStorage option. HCatalog is providing the glue that lets your Pig script interoperate with your Hive tables. You are then ordering the data by column1. And when you store the results, again, you are storing it in another table. These Pig scripts that are running never need to know where on the file system these files are actually being stored, and it has to know very little about schema about the data. All it needs to know is that there is a column there that it needs to sort by.

Please have a look at this basic 10 minutes video on HCatalog:

http://www.youtube.com/watch?v=_dVlNu4lqpE

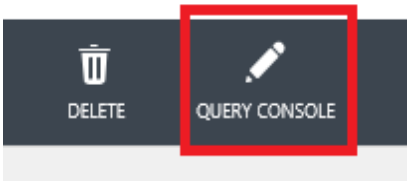
and this 27 minutes introduction video on HCatalog:

http://www.youtube.com/watch?v=6kwqEPgWkMI

HCatalog in HDInsight

Let's have a look at and see how HCatalog can be used to do that type of things.

1. Go to the HDInsight cluster remote desktop
2. Bring up the Hadoop Command Line tool
3. To get the sample data for this example, click on the query console from your cluster in Azure subscription



4. Enter the admin credentials for the cluster, which then logs you into the query console. It also adds the example data that you'll use for this demo.
5. Let's look at the data that was created in step 3 which you are going to work on:

```
C:\>hadoop fs -ls /HdiSamples/WebsiteLogSampleData/SampleLog
Found 1 items
-rwxrwxrwx   1     273431 2014-11-17 06:43 /HdiSamples/WebsiteLogSampleData/Samp
leLog/909f2b.log
```

This is the IIS web server log file. It has all the data traffic information. And in this log there are a few header rows that are prefixed with hash symbols and then there is some data, basically one line for each web request that was made on that day with details of the IP address, amount of data sent and received, and so on.

6. Now you would like to create couple of Hive tables that you are going to use to process this log with. And to do that, you are going to use something called HCatalog. There is a system variable, called HCAT_HOME which points to the location where whatever version of HCatalog is installed, is actually located on the file system. This HCAT_HOME is pointing towards the version specific installation.

   Under this HCAT_HOME folder, there is a folder called bin, under which is a script file called hcat.py. And there are number of things you can do with HCatalog including running Hive scripts and Hive commands. Here, you are going to run a file (CreateTables.hcatalog) that contains Hive commands.

```
C:\>%HCAT_HOME%\bin\hcat.py -f c:\scripts\CreateTables.hcatalog
OK
Time taken: 1.82 seconds
OK
Time taken: 0.5 seconds
```

   This CreateTables.hcatalog file has the following contents:

```
create table stagedlogs
(s_date string, s_time string, s_sitename string, cs_method string, cs_uristem string,
cs_uriquery string, s_port int, cs_username string, c_ip string, cs_useragent string,
cs_cookie string, cs_referer string, cs_host string, sc_status int, sc_substatus int,
sc_win32status int, sc_bytes int, cs_bytes int, s_timetaken int)
ROW FORMAT DELIMITED FIELDS TERMINATED BY ' ';

create table logdata
(s_date string, s_time string, s_sitename string, cs_method string, cs_uristem string,
cs_uriquery string, s_port int, cs_username string, c_ip string, cs_useragent string,
cs_cookie string, cs_referer string, cs_host string, sc_status int, sc_substatus int,
sc_win32status int, sc_bytes int, cs_bytes int, s_timetaken int)
ROW FORMAT DELIMITED FIELDS TERMINATED BY ' ';
```

   It is some regular HiveQL. It is creating a table called stagedlogs that has the schema from the IIS log file. You haven't specified a file location for this table, so it is going to be put under the default location for Hive (which happens to be /hive/warehouse/stagedlogs).

And then you have another table called logdata, with exactly the same schema and it will end up in the default location too. But this table is going to be used, when you remove the header information, and you load the remaining rows in the logdata table so that you have the clean set of data to work with.

When you run the hcat.py file, off it goes and starts executing the commands that are in the script file. You are now set up and ready to start operating on the data.

7. What you are going to do now is to load the data that you've staged into the table that you've just created and you'll use Hive to do that.

```
C:\>%hive_home%\bin\hive
```

8. Let's run the necessary code that just load data. This is the only time, when you as the person who is running the job, need to know what the path is on the HDFS system. It is the only time that path is hard-coded anywhere. You are taking the data that was staged into that folder and loading it into the stagedlogs table.

```
hive> LOAD DATA INPATH '/HdiSamples/WebsiteLogSampleData/SampleLog' INTO TABLE s
tagedlogs;
```

It moves the log file from this hard coded folder path to the location of the table which you don't need to be aware of. Hive is taking care of that folder path. At this point, you now got a table that contains raw data for us to work with.

9. Exit the Hive environment

```
hive> quit;
```

10. Now let's use Pig to actually cleanse the data, so you are not going to do the whole thing in Hive. You'll do some of the data operations in Pig and will use a Pig script for this called CleanAndLoadLogData.pig.

```
StagedLogs = LOAD 'stagedlogs' USING org.apache.hcatalog.pig.HCatLoader();

CleanLogs = FILTER StagedLogs BY SUBSTRING(s_date, 0, 1) != '#';

SortedLogs = ORDER CleanLogs BY s_date ASC;

STORE SortedLogs INTO 'logdata' USING org.apache.hcatalog.pig.HCatStorer();
```

Here, you are creating a relation called StagedLogs. And what you are doing is loading the data from the Hive table stagedlogs using HCatalog. The Pig script doesn't need to know where on the HDFS file system that data is. It just says that it knows the table stagedlogs and it needs the data from that.

The next statement starts filtering and it removes any row that starts with a hash symbol. Again, what's interesting here is that Pig doesn't need to know anything about the schema other than there is a column called s_date. If at some point later on you change the schema of this table by adding additional column or remove the columns or whatever, as long as you don't remove the s_date column, this Pig script will still continue to work. It still will go and filter on that column.

Then it sorts on that s_date column as well.

And then finally it takes the resulting SortedLogs relation generated and loads that into the logdata table. It doesn't need to know where on the file system to write the data to. It just loads into that table and HCatalog figures out where it needs to put the data.

You've abstracted the storage from Pig using HCatalog which is now enabling us to write a much more flexible and reusable Pig script.

11. Let's run the Pig engine and flag it with useHCatalog so that it knows it has to use HCatalog to interact with Hive. And then simply point it at the above script file that you just saw.

```
C:\>%PIG_HOME%\bin\pig -useHCatalog c:\scripts\CleanAndLoadLogData.pig
```

And it is going to run a Map/Reduce job. And you can see from the output the number of records read from stagedlogs and number of records stored into logdata.

```
Input(s):
Successfully read 648 records from: "stagedlogs"

Output(s):
Successfully stored 646 records in: "logdata"
```

12. If you now go and look in the Hive environment and see how many records there are in logdata by executing the commands shown here,

```
C:\>%HIVE_HOME%\bin\hive
```

```
hive> SELECT COUNT(*) FROM logdata;
```

```
OK
646
Time taken: 39.231 seconds, Fetched: 1 row(s)
hive>
```

It will go and run a Map/Reduce job, and there you can see for sure enough the row count in logdata table. The processing worked and in the entire process to do that, at only one point did you have to specify a hard-coded file path. Everything was just managed automatically for us and HCatalog provided that glue that enabled Pig and Hive to work together to do the problem solving for us.