Tutorial on Apache Hive

Table Of Contents

Introduction

Hive provides a means of running MapReduce job through an SQL-like scripting language, called HiveQL, that can be applied towards summarization, querying, and analysis of large volumes of data. HiveQL enables anyone already familiar with SQL to query the data. (While Pig is intuitive and easy-to-understand, it does require learning a new scripting language.)

In this tutorial, you will use a log file as input, and use HiveQL to query the data and report basic statistics. Our input file consists of a semi-structured log4j file in the following format

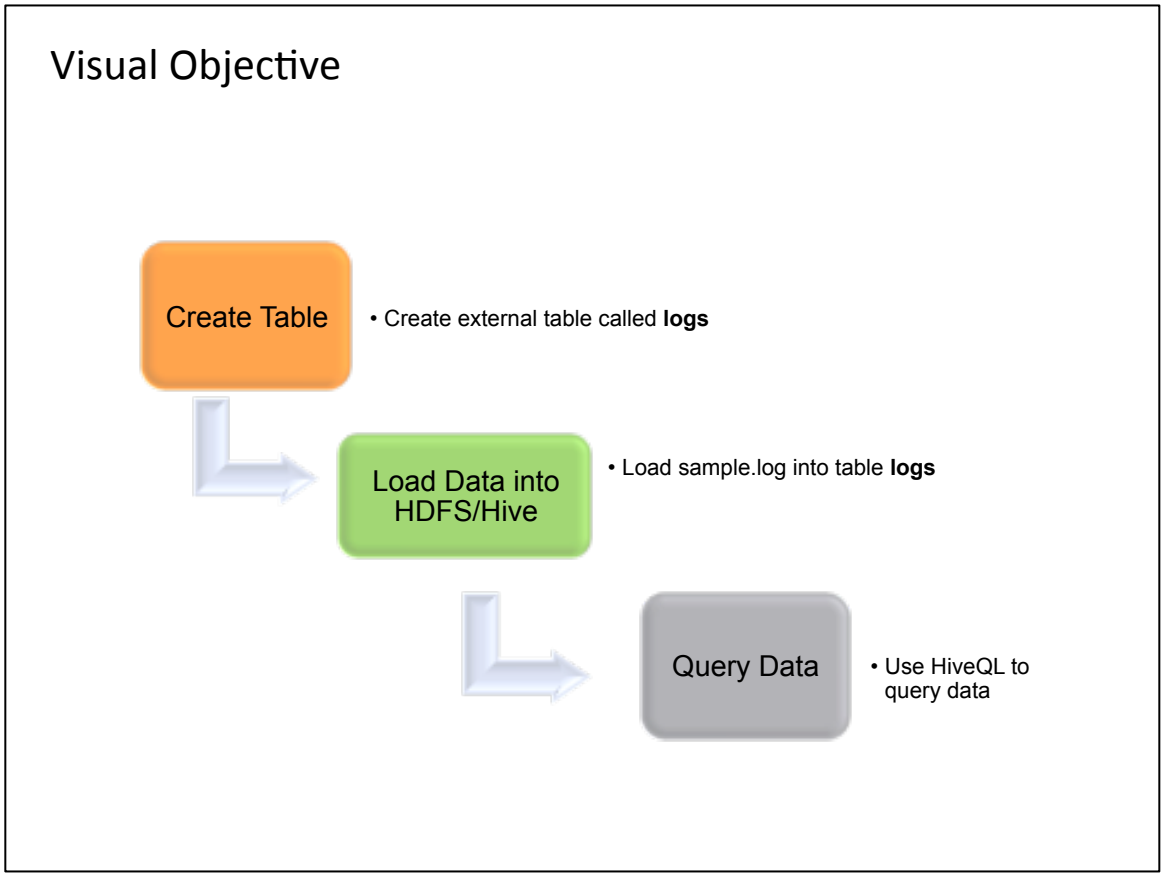
```
. . . . . .

2012-02-03   20:26:41   SampleClass3   [TRACE]   verbose detail
for id 1527353937
2012-02-03   20:26:41   SampleClass2   [TRACE]   verbose detail
for id 191364434
2012-02-03   20:26:41   SampleClass1   [DEBUG]   detail for id
903114158
2012-02-03   20:26:41   SampleClass8   [TRACE]   verbose detail
for id 1331132178
2012-02-03   20:26:41   SampleClass8   [INFO]   everything normal
for id 1490351510
2012-02-03   20:32:47   SampleClass8   [TRACE]   verbose detail
for id 1700820764
2012-02-03   20:32:47   SampleClass2   [DEBUG]   detail for id
364472047
2012-02-03   21:05:21   SampleClass6   [FATAL]   system problem
at id 1620503499
. . . . . .
```

The output data will be put into a file showing the various log4j log levels along with its frequency occurrence in our input file. A sample of these metrics is displayed below:

```
[TRACE] 8
[DEBUG] 4
[INFO]  1
[WARN]  1
[ERROR] 1
[FATAL] 1
```

This tutorial takes about 30 minutes to complete and is divided into the following tasks:

The visual representation of what you will accomplish in this tutorial is shown in the figure.



The Use Case

Databases are great for small sets of data and low latency queries. However, when it comes to **Big Data** and large data sets in terabytes, traditional SQL databases are not the ideal solution. Traditionally, database administrators have relied on scaling up by buying bigger hardware as database load increases and performance degrades.

Hive solves these problems by allowing users to scale out when querying **Big Data**. **Hive** queries data in parallel across multiple nodes using **MapReduce**, distributing the database across multiple hosts as load increases.

Hive can also be used as an alternative to writing java **MapReduce** jobs, because it provides an SQL-like interface to run complex queries against **Big Data**. By providing a simple, SQL like wrapper, complex **MapReduce** code can be avoided with a few lines of **SQL**-like entries.

Hive also allows programmers who are familiar with the MapReduce framework to be able to plug in their custom mappers and reducers to perform more sophisticated analysis that may not be supported by the built-in capabilities of the language.

Hive is best suited for batch processing of large amounts of immutable data (such as web logs). It is not appropriate for transaction applications that need very fast response times, such as database management systems. Hive is optimized for scalability (more machines can be added dynamically to the Hadoop cluster), extensibility (with MapReduce framework and other programming interfaces), and fault-tolerance. Latency is not a key design consideration.

Generally, all applications save errors, exceptions and other coded issues in a log file, so administrators can review the problems, or generate certain metrics from the log file data. These log files usually get quite large in size, containing a wealth of data that must be processed and mined.

Log files are therefore a good example of **big data**. Working with big data is difficult using relational databases and statistics/visualization packages. Due to the large amounts of data and the computation of this data, parallel software running on tens, hundreds, or even thousands of servers is often required to compute this data in a reasonable time. **Hadoop** provides a **Hive** data warehouse system that facilitates easy data summarization, ad-hoc queries, and the analysis of large datasets stored in Hadoop compatible file systems.

Pre-Requisites

Ensure that these pre-requisites have been met prior to starting the tutorial.

- Access to Hortonworks Virtual Sandbox—This tutorial uses a hosted solution that runs in an Amazon Web Services (AWS) EC2 environment. It provides a packaged environment for demonstration and trial use of the Apache Hadoop ecosystem on a single node (pseudo-distributed mode).

- The Virtual Sandbox is accessible as an Amazon Machine Image (AMI) and requires that you have an account with AWS.

- Working knowledge of Linux OS.

- Basic knowledge of MapReduce is recommended including having completed the Hortonworks Tutorial on Hadoop HDFS and MapReduce

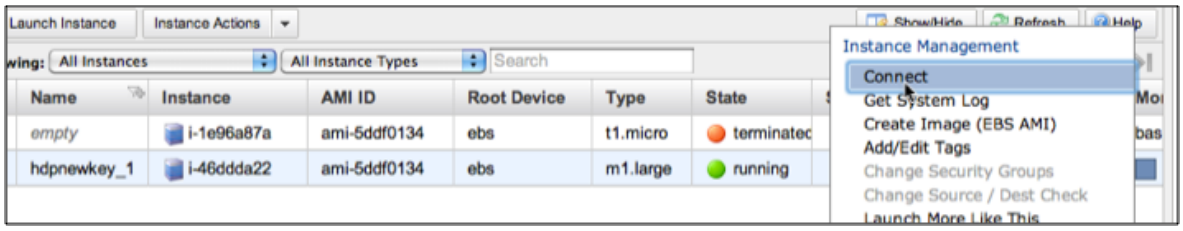
For help in getting an AWS account and configuring to the Hortonworks Virtual Sandbox, refer to Using the Hortonworks Virtual Sandbox.

Task 1: Access Your Hortonworks Virtual Sandbox

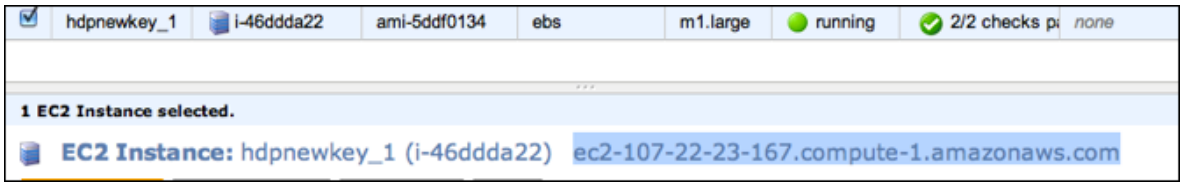
Note: If you have the Hortonworks Virtual Sandbox up and running, are connected to the instance via SSH, and verified that HDP services have been started, you can skip to Task 2.

Step 1: If you have not done so, access your AWS Management console. (https://console.aws.amazon.com/console/home and choose **EC2**)

On the AWS Management Console, go to the **My Instances** page. Right click on the row containing your instance and click **Connect**.



Step 2: Copy the AWS name of your instance.



Step 3: Connect to the instance using SSH. From the SSH client on your local client machine, enter:

```
ssh -i <Full_Path_To_Key_Pair_File> root@<EC2 Instance Name>
```

In the example above you would enter: ec2-107-22-23-167.compute-1.amazonaws.com.

Step 4: Verify that HDP services are started. MapReduce requires several services that should have been started if you completed the pre-requisite "Using The Hortonworks Virtual Sandbox".

These services are:

- NameNode
- JobTracker
- SecondaryNameNode
- DataNode
- TaskTracker

View the last 20 lines of the **hdp-stack-start-<date>-<time>.log** file (located here: /root). (Use **ls** to get the actual <date><time>) to verify required

```
# tail -20  hdp-stack-start-<date><time>
```

This file provides a list of all the Hadoop services that have started successfully. For example, the following screenshot provides the output of the tail end of this log file:

```
*****************          Java Process            ***************
2807 hdfs -Dproc_namenode
3135 hdfs -Dproc_secondarynamenode
3349 hdfs -Dproc_datanode
6289 mapred -Dproc_jobtracker
6581 mapred -Dproc_historyserver
6784 mapred -Dproc_tasktracker
7077 hcat -Dproc_jar
7296 oozie -Djava.util.logging.config.file=/var/lib/oozie/oozie-server/c
onf/logging.properties
```

If the services have not been started, use this command to start HDP services is **/etc/init.d/hdp-stack start.**

Task 2: Prep for Hive Processing

In the rest of the tutorial, you will create an table, load (sample.log) data into the table, and run various Hive queries on the data. The user "hcat" owns anything that you create in Hive so if you create the table as root user, you will not have proper permission to execute queries as root user. So you will make the sample.log file available to hcat user, and switch to hcat user.

Step 1: Change to the directory containing the tutorial:

```
# cd ~/tutorial
```

Step 2: Verify that the **sample.log** input file exists:

```
# ls
```

Step 3: Make **sample.log** file available to hcat user:

```
# cp sample.log /home/hcat/sample.log
```

Step 4: Switch from root to hcat superuser:

```
# su - hcat
```

Notice the prompt change from root to hcat.

Step 5: Verify sample.log file exists to hcat:

```
# ls
```

```
[hcat@hortonworks-sandbox ~]$ ls
sample.log
```

Step 6: Review the data in sample.log file:

```
# cat sample.log
```

Notice that the screen output below shows that the data follows a particular structure (except for the row that starts with "java.lang.Exception…").

```
2012-02-05 18:57:10 SampleClass6 [TRACE] verbose detail for id 637930791
2012-02-05 18:57:10 SampleClass3 [DEBUG] detail for id 1330329365
2012-02-05 18:57:10 SampleClass4 [TRACE] verbose detail for id 1956756252
2012-02-05 18:57:10 SampleClass9 [TRACE] verbose detail for id 522792294
2012-02-05 18:57:10 SampleClass5 [DEBUG] detail for id 2087594611
2012-02-05 18:57:10 SampleClass8 [INFO] everything normal for id 330067906
2012-02-05 18:57:10 SampleClass2 [TRACE] verbose detail for id 637230355
2012-02-05 18:57:10 SampleClass7 [TRACE] verbose detail for id 1518005850
2012-02-05 18:57:10 SampleClass4 [DEBUG] detail for id 1507936393
2012-02-05 18:57:10 SampleClass5 [TRACE] verbose detail for id 1920127698
2012-02-05 18:57:10 SampleClass3 [TRACE] verbose detail for id 1485023437

java.lang.Exception: 2012-02-05 18:57:10 SampleClass7 [ERROR] incorrect format for id 827562608
        at com.osa.mocklogger.MockLogger$2.run(MockLogger.java:83)
2012-02-05 18:57:10 SampleClass2 [DEBUG] detail for id 166977413
2012-02-05 18:57:10 SampleClass1 [TRACE] verbose detail for id 1803029938
2012-02-05 18:57:10 SampleClass3 [TRACE] verbose detail for id 932077821
2012-02-05 18:57:10 SampleClass3 [DEBUG] detail for id 1357054067
2012-02-05 18:57:10 SampleClass2 [TRACE] verbose detail for id 636605754
2012-02-05 18:57:10 SampleClass1 [TRACE] verbose detail for id 1613557382
2012-02-05 18:57:10 SampleClass5 [DEBUG] detail for id 1069370972
2012-02-05 18:57:10 SampleClass1 [TRACE] verbose detail for id 590508561
```

Starting from left to right, the structured data rows have a date in column 1, timestamp in column 2, class name in column 3, severity in column 4, and so on.

The row starting with "java.lang.Exception" does not follow this "well-formed" data structure and is therefore, considered unstructured. The following table shows the key differences between the structured rows and unstructured rows.

Data Type	Date Column	Severity Column
Structured	1	4
Unstructured	2	5

Task 3: Create Hive Table and Load Data

In this task, you will start hive, create an external table, and load sample.log data into the table.

Step 1: Start **Hive** by executing:

```
# hive
```

```
[hcat@hortonworks-sandbox ~]$ hive
WARNING: org.apache.hadoop.metrics.jvm.EventCounter is deprecated. Please use org.apache.hadoo
p.log.metrics.EventCounter in all the log4j.properties files.
Logging initialized using configuration in jar:file:/usr/share/hcatalog/lib/hive-common-0.4.0.
jar!/hive-log4j.properties
Hive history file=/tmp/hcat/hive_job_log_hcat_201204181719_1563921355.txt
hive>
```

Step 2: Create a table called "logs" in Hive to store sample data:

```
hive> CREATE TABLE logs(t1 string, t2 string, t3 string, t4
string, t5 string, t6 string, t7 string) ROW FORMAT DELIMITED
FIELDS TERMINATED BY ' ';
```

Note: The command is terminated by two single quotes with a space in between.

Step 3: Load the sample.log data into the logs table you just created:

```
hive> LOAD DATA LOCAL INPATH 'sample.log' OVERWRITE INTO TABLE
logs;
```

Task 4: Run Hive Queries

In this task, you will run simple Hive queries on the data.

Step 1: Run the following query to return the count of the different severity levels from the structured data:

```
hive> SELECT t4 AS sev, COUNT(*) AS cnt FROM
```

```
logs WHERE t4 LIKE '[%' GROUP BY t4;
```

```
[DEBUG] 15608
[ERROR] 94
[FATAL] 19
[INFO]  3355
[TRACE] 29950
[WARN]  178
```

Step 2: Run the following query to return the count of the different severity levels from the unstructured data:

```
hive> SELECT t5 AS sev, COUNT(*) AS cnt FROM logs WHERE
t5 LIKE '[%' GROUP BY t5;
```

```
[ERROR] 87
[FATAL] 18
[WARN]  183
```

Step 3: Now count the results by adding the output of both the queries:

```
hive> SELECT  L.sev, SUM(L.cnt) FROM (
   SELECT t4 AS sev, COUNT(*) AS cnt FROM logs WHERE t4 LIKE
'[%' GROUP BY t4
    UNION ALL
    SELECT t5 AS sev, COUNT(*) AS cnt FROM logs WHERE t5 LIKE
'[%' GROUP BY t5
    ) L GROUP BY L.sev;
```

The output combines the results from the structured and unstructured data. Also notice also that three MapReduce jobs were launched for this operation.

```
MapReduce Jobs Launched:
Job 0: Map: 1  Reduce: 1   Accumulative
Job 1: Map: 1  Reduce: 1   Accumulative
Job 2: Map: 1  Reduce: 1   Accumulative
Total MapReduce CPU Time Spent: 15 seco
OK
[DEBUG] 15608
[ERROR] 181
[FATAL] 37
[INFO]  3355
[TRACE] 29950
[WARN]  361
```

Task 5: Tutorial Clean Up

The clean up task applies to this tutorial only; it is not necessarily performed in an actual deployment. In this task, you will delete the table and the data so that if you like, you can run the tutorial again.

Step 1: Delete the table logs:

```
hive> drop table logs;
```

Step 2: Quit hive:

```
hive> quit;
```

Step 3: Go back to root user:

```
hcat@hortonworks-sandbox ~]$ exit
```

Congratulations! You have successfully completed this tutorial.

Additional Resources

For the latest Hive documentation from Apache Hadoop project, visit:
http://hive.apache.org/

For more information on an array services to support your implementation, visit:
http://hortonworks.com/support/

For Hadoop training and certification for developers or administrators, visit:
http://hortonworks.com/training/