# Lab1: Getting Started with HDInsight

## Create an HDInsight Cluster on Azure

## Connecting to an HDInsight Cluster using SSH client

## Browse Cluster Storage

1. In the SSH console, enter the following command to view the contents of the root folder in the HDFS file system.

**hdfs dfs -ls /**

1. Enter the following command to view the contents of the /example folder in the HDFS file system. This folder contains subfolders for sample apps, data, and JAR components.

**hdfs dfs -ls /example**

1. Enter the following command to view the contents of the /example/data/gutenberg folder, which contains sample text files:

**hdfs dfs -ls /example/data/gutenberg**

1. Enter the following command to view the text in the davinci.txt file:

**hdfs dfs -text /example/data/gutenberg/davinci.txt**

## Run a MapReduce Jab

1. Enter the following command to view the sample Java jars stored in the cluster head node:

**ls /usr/hdp/current/hadoop-mapreduce-client**

1. Enter the following command on a single line to get a list of MapReduce functions in the hadoop-mapreduce-examples.jar:

**hadoop jar /usr/hdp/current/hadoop-mapreduce-client/hadoop-mapreduce-examples.jar**

1. Enter the following command on a single line to get help for the wordcount function in the hadoop-mapreduce-examples.jar that is stored in the cluster head:

**hadoop jar /usr/hdp/current/hadoop-mapreduce-client/hadoop-mapreduce-examples.jar wordcount**

1. Enter the following command on a single line to run a MapReduce job using the wordcount function in the hadoop-mapreduce-examples.jar jar to process the davinci.txt file you viewed earlier and store the results of the job in the /example/results folder:

**hadoop jar /usr/hdp/current/hadoop-mapreduce-client/hadoop-mapreduce-examples.jar wordcount /example/data/gutenberg/davinci.txt /example/results**

1. Wait for the MapReduce job to complete, and then enter the following command to view the output folder, and note that a file named part-r-00000 has been created by the job.

**hdfs dfs -ls /example/results**

1. Enter the following command to view the results in the output file:

**hdfs dfs -text /example/results/part-r-00000**

## Upload a File to Azure Blob Storage

1. Open up the Microsoft Azure Storage Explorer and create a New Folder (+) named reviews. Drag and drop the **Lab01/reviews.txt** file on the new folder.
2. Refresh the Azure Storage Explorer to check the file copied properly. The Azure Storage Explorer will create a virtual directory and won’t generate a folder until it includes some data in the folder.

## Process the Uploaded Data

1. Switch to the SSH console for your HDInsight cluster, and enter the following command on a single line to run a MapReduce job using the wordcount function in the hadoop-mapreduceexamples.jar jar to process the reviews.txt file you uploaded and store the results of the job in the /reviews/results folder:

**hadoop jar /usr/hdp/current/hadoop-mapreduce-client/hadoop-mapreduce-examples.jar wordcount /reviews/reviews.txt /reviews/results**

1. Wait for the MapReduce job to complete, and then enter the following command to view the output folder, and verify that a file named part-r-00000 has been created by the job.

**hdfs dfs -ls /reviews/results**

## Download the Results

# Lab2: Processing Big Data with Hive

## Upload the Log Files to Azure Storage

1. Open up the Microsoft Azure Storage Explorer and create a **data** folder. Then, upload the Lab02/logs folder.

## View the Uploaded Data Files

1. In the SSH console for your cluster, enter the following command to view the contents of the /data/logs folder in the HDFS file system.

**hdfs dfs -ls /data/logs**

1. Verify that the folder contains six log files

## Create a Hive Table for the Raw Log Data

1. In the SSH console for your cluster, enter the following command to start the Hive command line interface:

**hive**

1. In the Hive command line interface, enter the following HiveQL statement to create a table named rawlogs:

**CREATE TABLE rawlog (**

**log\_date STRING,**

**log\_time STRING,**

**c\_ip STRING,**

**cs\_username STRING,**

**s\_ip STRING,**

**s\_port STRING,**

**cs\_method STRING,**

**cs\_uri\_stem STRING,**

**cs\_uri\_query STRING,**

**sc\_status STRING,**

**sc\_bytes INT,**

**cs\_bytes INT,**

**time\_taken INT,**

**cs\_user\_agent STRING,**

**cs\_referrer STRING)**

**ROW FORMAT DELIMITED FIELDS TERMINATED BY ' ';**

1. Wait for the job to complete.
2. Enter the following command to verify that the table has been created (there may also be an existing table named hivesampletable):

**SHOW TABLES;**

1. Enter the following command to query the table, and verify that no rows are returned:

**SELECT \* FROM rawlog;**

## Load the Source Data into the Raw Log Table

1. In the Hive command line interface, enter the following HiveQL statement to move the log files you previously uploaded into the folder for the rawlogs table

**LOAD DATA INPATH '/data/logs' INTO TABLE rawlog;**

1. Wait for the job to complete.
2. Enter the following command to return the first 100 rows in the rawlogs table:

**SELECT \* FROM rawlog LIMIT 100;**

1. View the output, noting that the query retrieved the first 100 rows from the table in tab-delimited text format. Note that the output includes rows containing comments from the source document (the first column value for these rows is prefixed with a # character, and null values are used for columns in the table schema for which there are no values in the source data).

## Clean the Log Data

1. In the Hive command line interface, enter the following HiveQL statement to create an external table named cleanlog based on the /data/cleanlog folder:

**CREATE EXTERNAL TABLE cleanlog (**

**log\_date DATE,**

**log\_time STRING,**

**c\_ip STRING,**

**cs\_username STRING,**

**s\_ip STRING,**

**s\_port STRING,**

**cs\_method STRING,**

**cs\_uri\_stem STRING,**

**cs\_uri\_query STRING,**

**sc\_status STRING,**

**sc\_bytes INT,**

**cs\_bytes INT,**

**time\_taken INT,**

**cs\_user\_agent STRING,**

**cs\_referrer STRING)**

**ROW FORMAT DELIMITED FIELDS TERMINATED BY ' '**

**STORED AS TEXTFILE LOCATION '/data/cleanlog';**

1. Wait for the job to complete.
2. Enter the following command to extract rows that do not contain comments from the rawlogs table, and insert them into the cleanlog table:

**INSERT INTO TABLE cleanlog**

**SELECT \* FROM rawlog**

**WHERE SUBSTR(log\_date, 1, 1) <> '#';**

1. Wait for the job to complete.
2. Enter the following command to retrieve the first 100 rows from the cleanlog table:

**SELECT \* FROM cleanlog LIMIT 100;**

1. View the results returned by the query, noting that the cleanlog table does not include any rows prefixed with a # character.

## Create a View

1. In the Hive command line interface, enter the following command to create a view named vDailySummary:

**CREATE VIEW vDailySummary**

**AS**

**SELECT log\_date, COUNT(\*) AS requests,**

**SUM(cs\_bytes) AS inbound\_bytes,**

**SUM(sc\_bytes) AS outbound\_bytes**

**FROM cleanlog**

**GROUP BY log\_date;**

1. Wait for the job to complete.
2. Enter the following command to query the vDailySummary view:

**SELECT \* FROM vDailySummary ORDER BY log\_date;**

1. Wait for the job to complete and then view the output returned by the query.

## Create and Load a Partitioned Table

1. In the Hive command line interface, enter the following command to create a partitioned table:

**CREATE EXTERNAL TABLE partitionedlog (**

**log\_day int,**

**log\_time STRING,**

**c\_ip STRING,**

**cs\_username STRING,**

**s\_ip STRING,**

**s\_port STRING,**

**cs\_method STRING,**

**cs\_uri\_stem STRING,**

**cs\_uri\_query STRING,**

**sc\_status STRING,**

**sc\_bytes INT,**

**cs\_bytes INT,**

**time\_taken INT,**

**cs\_user\_agent STRING,**

**cs\_referrer STRING)**

**PARTITIONED BY (log\_year int, log\_month int)**

**ROW FORMAT DELIMITED FIELDS TERMINATED BY ' '**

**STORED AS TEXTFILE LOCATION '/data/partitionedlog';**

1. Enter the following command to load data from the rawlog table into the partitioned table:

**SET hive.exec.dynamic.partition.mode=nonstrict;**

**SET hive.exec.dynamic.partition = true;**

**INSERT INTO TABLE partitionedlog PARTITION(log\_year, log\_month)**

**SELECT DAY(log\_date),**

**log\_time,**

**c\_ip,**

**cs\_username,**

**s\_ip,**

**s\_port,**

**cs\_method,**

**cs\_uri\_stem,**

**cs\_uri\_query,**

**sc\_status,**

**sc\_bytes,**

**cs\_bytes,**

**time\_taken,**

**cs\_user\_agent,**

**cs\_referrer,**

**YEAR(log\_date),**

**MONTH(log\_date)**

**FROM rawlog**

**WHERE SUBSTR(log\_date, 1, 1) <> '#';**

1. Wait for the query job complete. By partitioning the data in this way, queries filtered on year and month will benefit from having to search a smaller volume of data to retrieve the results.
2. Enter the following command to query the partitioned table, and when the job has completed, view the results:

**SELECT log\_day, count(\*) AS page\_hits**

**FROM partitionedlog**

**WHERE log\_year=2008 AND log\_month=6**

**GROUP BY log\_day;**

## View Folder for a Partitioned Table

1. In the Hive command line interface, enter the following command to exit Hive:

**exit;**

1. At the command prompt, enter the following command to view the /data/partitionedlog folder:

**hdfs dfs -ls /data/partitionedlog**

1. Note that the folder contains a subfolder for each value in the first partitioning key (in this case, a single folder named log\_year=2008).
2. Enter the following command to view the contents of the log\_year=2008 subfolder (note that quotation marks are required because the folder name includes a “=” character):

**hdfs dfs -ls "/data/partitionedlog/log\_year=2008"**

1. Note that the subfolder contains a further subfolder for each value in the second partitioning key (in this case, a folder for each log\_month value in the source data).
2. Enter the following command to view the contents of the log\_month=6 subfolder:

**hdfs dfs -ls "/data/partitionedlog/log\_year=2008/log\_month=6"**

1. Note that the subfolders for each month contain one or more data files. In this case, a file with a name similar to 000000\_0 contains all of the log records with a year of 2008 and a month of 6.
2. Enter the following command to view the last few kilobytes of data in the file (modifying the file name if necessary):

**hdfs dfs -tail "/data/partitionedlog/log\_year=2008/log\_month=6/000001\_0"**

1. Note that the first field in the data file contains the day of the month (the last few rows will contain the value 30.) The year and month are not stored in the data file, but are determined from the partitioning subfolder in which the file is stored - this file contains only the records for June 2008.
2. Close all open command windows and Azure Storage Explorer.

# Lab 3: Processing Data with Pig

## View and Upload the Source Data

1. Use a text editor to view the **heathrow.txt** file in the Lab03 folder where you extracted the lab files for this course. Note that the file contains monthly weather observation data for Heathrow airport in London from 1948 to 2015. The first few lines of the file contain unstructured text, which is followed by a multiple rows of tab-delimited values. Note: The file contains public sector information licensed under the Open Government License.
2. Close the text file without saving any changes.
3. Start Azure Storage Explorer, and if you are not already signed in, sign into your Azure subscription.
4. Expand your storage account and the Blob Containers folder, and then double-click the blob container for your HDInsight cluster.
5. In the Upload drop-down list, click Upload Files. Then upload heathrow.txt as a block blob to a new folder named data/weather in root of the container.
6. Keep Azure Storage Explorer open – you will use it again in a later procedure.

## Connect to the Cluster

## View the Uploaded Data File

1. In the SSH console window, enter the following command to view the contents of the **/data/weather** folder in the HDFS file system.

**hdfs dfs -ls /data/weather**

1. Verify that the folder contains the heathrow.txt file you uploaded.

## Use the Grunt Shell to Run Pig Commands

1. In the console containing an SSH session to your cluster, enter the following command to start the Grunt shell:

**pig**

1. In the Grunt shell, enter the following Pig Latin statement to load the files in the /data/weather folder into a relation called Source. The data is loaded into a tab-delimited schema with seven columns. Lines with no tab characters will result in a row (or tuple) containing a single value in the first column and NULL values in the remaining columns.

**Source = LOAD '/data/weather' USING PigStorage('\t') AS (year:chararray, month:chararray, maxtemp:float, mintemp:float, frost:int, rainfall:float, sunshine:float);**

1. Ignore any warning messages, and enter the following Pig Latin statement to filter the data so that only tuples with a value in the maxtemp and mintemp columns are included.

**Data = FILTER Source BY maxtemp IS NOT NULL AND mintemp IS NOT NULL;**

1. Enter the following Pig Latin statement to further filter the data to remove the header row.

**Readings = FILTER Data BY year != 'yyyy';**

1. Enter the following Pig Latin statement to group the data by year.

**YearGroups = GROUP Readings BY year;**

1. Enter the following Pig Latin statement to calculate average maxtemp and average mintemp for each year.

**AggTemps = FOREACH YearGroups GENERATE group AS year, AVG(Readings.maxtemp) AS avghigh, AVG(Readings.mintemp) AS avglow;**

1. Enter the following Pig Latin statement to sort the temperature data by year.

**SortedResults = ORDER AggTemps BY year;**

1. Enter the following Pig Latin statement to display the results in the console. This runs a MapReduce job to perform all of the transformations you have entered so far.

**DUMP SortedResults;**

1. Wait for the job to complete, and then view the results, which include the average monthly maximum and minimum temperatures for each year. Then enter the following command to exit Pig.

**quit;**

1. Keep the SSH console open, you will return to it in the next procedure

## Run a Pig Latin Script

1. Use a text editor to open the **scrub\_weather.pig** file in the folder.
2. Review the Pig Latin code in this file, and note that the script performs the following tasks:
   1. Loads the tab-delimited data in the **/data/weather** folder into a schema that includes the columns year, month, maxtemp, mintemp, frostdays, rainfall, and sunshinehours.
   2. Filters the data to remove text notes and header rows.
   3. Replaces any “---“ values (which indicate missing data) in the sunshinehours column with an empty string.
   4. Splits the data into a relation in which sunshinehours contains a “’#” character denoting a sensor reading (which makes the data dirty) and a relation in which sunshinehours is already clean.
   5. Cleans the rows in which sunshinehours contains a “#” by using the SUBSTRING and INDEXOF functions.
   6. Re-combines the cleaned rows with the rows that were already clean.
   7. Sorts the data by year and month.
   8. Stores the cleaned and sorted data in the /data/scrubbedweather folder as a spacedelimited text file.
3. Close the text editor.
4. In Azure Storage Explorer, in the Upload drop-down list, click Upload Files. Then upload **scrub\_weather.pig** as a block blob to the existing data folder.
5. Keep Azure Storage Explorer open – you will use it again in a later procedure.
6. Return to the SSH console for your cluster, and enter the following command to run the pig script you uploaded to Azure storage:

**pig wasb:///data/scrub\_weather.pig**

1. Wait for the script to finish, then enter the following command to verify that the script has written the results to the output folder:

**hdfs dfs -ls /data/scrubbedweather**

1. Enter the following command to view the cleaned weather data:

**hdfs dfs -text /data/scrubbedweather/part-r-00000**

1. Keep the SSH console for your cluster open, you will return to it later in the lab.

## Use a Python UDF from Pig

1. Use a text editor to open the **convert\_temp.py** file in the local Lab03 folder. Then review the Python code this file contains, and note that the code defines an output schema that includes a Pig bag structure named f\_readings. The bag contains fields named year, month, maxtemp, mintemp, frostdays, rainfall, and sunshinehours. The code then defines a function named fahrenheit with an input parameter named c\_reading. This function:
   1. Splits the input parameter into year, month, maxtemp, mintemp, frostdays, rainfall, and sunshinehours variables.
   2. Creates a variable named maxtemp\_f, which is calculated as maxtemp multiplied by nine divided by five and added to 32 (the equation to convert Celsius to Fahrenheit).
   3. Similarly, creates a variable named mintemp\_f with a value of mintemp converted to Fahrenheit.
   4. Returns the year, month, maxtemp\_f, mintemp\_f, frostdays, rainfall, and sunshinehours variables.
2. When you have finished viewing the code, close the text editor without saving any changes.
3. Use the text editor to open the **convert\_weather.pig** file in the local Lab03 folder. Then review the Pig Latin code this file contains, and note that it performs the following tasks:
   1. Registers the convert\_temp.py Python file as a Jython UDF.
   2. Loads the scrubbedweather source data you generated in the previous exercise into a relation named Source, with a single character array value for each line of text.
   3. Creates a relation named ConvertedReadings that uses the fahrenheit function in the **convert\_temp.py** file to generate each row.
   4. Stores the ConvertedReadings relation in the /data/convertedweather folder.
4. When you have finished viewing the code, close the text editor without saving any changes.
5. In Azure Storage Explorer, in the Upload drop-down list, click Upload Files. Then upload both convert\_temp.py and convert\_weather.pig as block blobs to the existing data folder.
6. Keep Azure Storage Explorer open – you will use it again in a later procedure.
7. Return to the SSH console for your cluster, and enter the following command to run the Pig script you uploaded to Azure storage:

**pig wasb:///data/convert\_weather.pig**

1. Wait for the script to finish, then enter the following command to verify that the script has written the results to a file named part-m-00000 in the output folder:

**hdfs dfs -ls /data/convertedweather**

1. Enter the following command to view the converted weather data:

**hdfs dfs -text /data/convertedweather/part-m-00000**

1. Keep the SSH console for your cluster open, you will return to it later in the lab.

## Use a Python UDF from Hive

1. Use a text editor to open the **create\_table.hql** file in the local folder. Then review the HiveQL code this file contains, and note that the code creates a table named weather based on the **/data/convertedweather** folder generated by the **convert\_weather.pig** Pig script in the previous procedure. The table includes a column named rainfall, in which the amount of rainfall is measured in millimeters.
2. When you have finished viewing the code, close the text editor without saving any changes.
3. Use the text editor to open the **convert\_rain.py** file in the local Lab03 folder. Then review the Python code this file contains, and note that it performs the following tasks:
   1. Reads each line of text input from the standard input port.
   2. Removes the new-line character in each line.
   3. Splits the remaining text into fields based on a tab-delimiter.
   4. Calculates a rainfall\_inches field by converting the value in the rainfall\_mm field to inches.
   5. Writes the fields to the standard output buffer, replacing rainfall\_mm with rainfall\_inches.
4. When you have finished viewing the code, close the text editor without saving any changes.
5. In Azure Storage Explorer, in the Upload drop-down list, click Upload Files. Then upload both **convert\_rain.py** and **create\_table.hql** as block blobs to the existing data folder.
6. Keep Azure Storage Explorer open – you will use it again in a later procedure.
7. Return to the SSH console for your cluster, and enter the following command to run the Hive script you uploaded to Azure storage:

**hive -i wasb:///data/create\_table.hql**

1. Wait for the script to open the Hive command line interface, and then enter the following query to verify that the weather table has been created on the output previously generated by Pig:

**SELECT \* FROM weather;**

1. Enter the following code to import the Python code you uploaded and use the UDF it defines to query the weather table:

**add file wasb:///data/convert\_rain.py;**

**SELECT TRANSFORM (year, month, max\_temp, min\_temp, frost\_days, rainfall, sunshine\_hours)**

**USING 'python convert\_rain.py' AS**

**(year INT, month INT, max\_temp FLOAT, min\_temp FLOAT, frost\_days INT, rainfall FLOAT, sunshine\_hours FLOAT)**

**FROM weather;**

1. View the results, noting that rainfall (the second-last column) contains values in inches instead of millimeters.
2. Enter the following command to exit Hive:

**exit;**

# Lab 4: Using HBase for NoSQL Data

## Create an HBase Table

* 1. In the console window for your SSH connection, enter the following command to start the HBase shell.

**hbase shell**

* 1. At the hbase prompt, enter the following command to create a table named Stocks with two column families named Current and Closing.

**create 'Stocks', 'Current', 'Closing'**

* 1. Enter the following command to insert a field for a record with the key ABC and a value of 97.3 for a column named Price in the Current column family.

**put 'Stocks', 'ABC', 'Current:Price', 97.3**

* 1. Enter the following command to insert a field for record ABC and a value of 95.7 for a column named Price in the Closing column family.

**put 'Stocks', 'ABC', 'Closing:Price', 95.7**

* 1. Enter the following command to return all rows from the table.

**scan 'Stocks'**

* 1. Verify that the output shows the two values you entered for the row ABC, as shown here:

**ROW COLUMN+CELL**

**ABC column=Closing:Price, timestamp=nnn, value=95.7**

**ABC column=Current:Price, timestamp=nnn, value=97.3**

* 1. Enter the following command to insert a field for record ABC and a value of Up for a column named Status in the Current column family.

**put 'Stocks', 'ABC', 'Current:Status', 'Up'**

* 1. Enter the following command to return the values for row ABC.

**get 'Stocks', 'ABC'**

* 1. Verify that the output shows the values of all cells for row ABC, as shown here:

**COLUMN CELL**

**Closing:Price timestamp=nnn, value=95.7**

**Current:Price timestamp=nnn, value=97.3**

**Current:Status timestamp=nnn, value=Up**

* 1. Enter the following command to set the Price column in the Current column family of row ABC to 99.1.

**put 'Stocks', 'ABC', 'Current:Price', 99.1**

* 1. Enter the following command to return the values for row ABC.

**get 'Stocks', 'ABC'**

* 1. Verify that the output shows the updated values of all cells for row ABC, as shown here:

**COLUMN CELL**

**Closing:Price timestamp=nnn, value=95.7**

**Current:Price timestamp=nnn, value=99.1**

**Current:Status timestamp=nnn, value=Up**

* 1. Note the timestamp value for the Current:Price cell. Then enter the following command to retrieve the previous version of the cell value by replacing nnn-1 with the timestamp for Current:Price minus 1 (for example, if the timestamp for Current:Price in the results above is 144012345678, replace nnn-1 with 144012345677.)

**get 'Stocks', 'ABC', {TIMERANGE=>[0,nnn-1]}**

* 1. Verify that the output shows previous Current:Price value, as shown here:

**COLUMN CELL**

**Closing:Price timestamp=nnn, value=95.7**

**Current:Price timestamp=nnn, value=97.3**

**Current:Status timestamp=nnn, value=Up**

* 1. Enter the following command to delete the Status column in the Current column family of row ABC.

**delete 'Stocks', 'ABC', 'Current:Status'**

* 1. Enter the following command to return the values for row ABC.

**get 'Stocks', 'ABC'**

* 1. Verify that the Current:Status cell has been deleted as shown here:

**COLUMN CELL**

**Closing:Price timestamp=nnn, value=95.7**

**Current:Price timestamp=nnn, value=99.1**

* 1. Enter the following command to exit the HBase shell and return to the Hadoop command line.

**quit**

* 1. Minimize the remote desktop window (you will return to the Hadoop Command Line later.)

## Bulk Load Data into on HBase Table

* 1. In the Labs folder where you extracted the lab files for this course on your local computer, in the Lab04 folder, open **stocks.txt** in a text editor. Note that this file contains tab-delimited records of closing and current prices for a variety of stocks. Then close the text editor without saving any changes.
  2. Start Azure Storage Explorer, and if you are not already signed in, sign into your Azure subscription.
  3. Expand your storage account and the Blob Containers folder, and then double-click the blob container for your HDInsight cluster.
  4. In the Upload drop-down list, click Upload Files. Then upload stocks.txt as a block blob to a new folder named data in root of the container.
  5. Switch back to the SSH connection console window, and enter the following command to verify that the file is now in the shared storage.

**hdfs dfs -ls /data**

* 1. In the SSH connection console, enter the following command (on a single line) to transform the tab-delimited stocks.txt data to the HBase StoreFile format.

**hbase org.apache.hadoop.hbase.mapreduce.ImportTsv - Dimporttsv.columns="HBASE\_ROW\_KEY,Closing:Price,Current:Price" - Dimporttsv.bulk.output="/data/storefile" Stocks /data/stocks.txt**

* 1. Wait for the MapReduce job to complete (this may take several minutes). Then enter the following command (on a single line) to load the transformed data into the Stocks table you created previously.

**hbase org.apache.hadoop.hbase.mapreduce.LoadIncrementalHFiles /data/storefile Stocks**

* 1. Wait for the MapReduce job to complete.

## Query the Bulk Loaded Data

1. Enter the following command to start the HBase shell.

**hbase shell**

1. Enter the following command to return all rows from the table.

**scan 'Stocks'**

1. Verify that the output includes rows for the ABC stock you entered previously and the stocks in the stocks.txt file you imported, as shown here:

**ROW COLUMN+CELL**

**AAA column=Closing:Price, timestamp=nnn, value=12.8**

**AAA column=Current:Price, timestamp=nnn, value=14.2**

**ABC column=Closing:Price, timestamp=nnn, value=95.7**

**ABC column=Current:Price, timestamp=nnn, value=99.1**

**BBB column=Closing:Price, timestamp=nnn, value=30.1**

**BBB column=Current:Price, timestamp=nnn, value=30.1**

**CBA column=Closing:Price, timestamp=nnn, value=120.3**

**CBA column=Current:Price, timestamp=nnn, value=120.3**

**GDM column=Closing:Price, timestamp=nnn, value=126.7**

**GDM column=Current:Price, timestamp=nnn, value=135.2**

...

1. Enter the following command to return only the Current:Price column for each row:

**scan 'Stocks', {COLUMNS => 'Current:Price'}**

1. Verify that the output includes a row for each stock with only the Current:Price column, as shown here:

**ROW COLUMN+CELL**

**AAA column=Current:Price, timestamp=nnn, value=14.2**

**ABC column=Current:Price, timestamp=nnn, value=99.1**

**BBB column=Current:Price, timestamp=nnn, value=30.1**

**CBA column=Current:Price, timestamp=nnn, value=120.3**

**GDM column=Current:Price, timestamp=nnn, value=135.2**

...

1. Enter the following command to return only the first three rows:

**scan 'Stocks', {LIMIT => 3}**

1. Verify that the output includes data for only three rows (there are two columns per row), as shown here:

**ROW COLUMN+CELL**

**AAA column=Closing:Price, timestamp=nnn, value=12.8**

**AAA column=Current:Price, timestamp=nnn, value=14.2**

**ABC column=Closing:Price, timestamp=nnn, value=95.7**

**ABC column=Current:Price, timestamp=nnn, value=99.1**

**BBB column=Closing:Price, timestamp=nnn, value=30.1**

**BBB column=Current:Price, timestamp=nnn, value=30.1**

1. Enter the following command to return only the rows for with key values between C and H:

**scan 'Stocks', {STARTROW=>'C', STOPROW=>'H'}**

1. Verify that the output includes only rows for stocks with stock codes between ‘C’ and ‘H’, as shown here:

**ROW COLUMN+CELL**

**CBA column=Closing:Price, timestamp=nnn, value=120.3**

**CBA column=Current:Price, timestamp=nnn, value=120.3**

**GDM column=Closing:Price, timestamp=nnn, value=126.7**

**GDM column=Current:Price, timestamp=nnn, value=135.2**

1. Enter the following command to exit the HBase shell:

**quit**

## Create a Hive Table on an HBase Table

1. In the SSH console for your cluster, enter the following command to start the Hive command line interface:

**hive**

1. At the Hive prompt, enter the following code to create a Hive table named StockPrices that is based on the Stocks HBase table:

**CREATE EXTERNAL TABLE StockPrices (**

**Stock STRING,**

**ClosingPrice FLOAT,**

**CurrentPrice FLOAT)**

**STORED BY 'org.apache.hadoop.hive.hbase.HBaseStorageHandler'**

**WITH SERDEPROPERTIES**

**('hbase.columns.mapping' = ':key,Closing:Price,Current:Price')**

**TBLPROPERTIES ('hbase.table.name' = 'Stocks');**

1. Wait for the table to be created.
2. Enter the following code to query the Hive table:

**SELECT Stock, CurrentPrice, ClosingPrice,**

**IF(CurrentPrice > ClosingPrice, 'Up',IF (CurrentPrice < ClosingPrice, 'Down', '-'))**

**AS Status**

**FROM StockPrices**

**ORDER BY Stock;**

1. Wait for the MapReduce job to complete, and then view the results. Note that the status for stock ABC is “Up” (because the current stock price is higher than the previous closing price).
2. Enter the following command to exit the Hive shell.

**quit;**

1. Enter the following command to start the HBase shell:

**hbase shell**

1. In the HBase shell, enter the following command to set the Price column in the Closing column family of row ABC to 92.8.

**put 'Stocks', 'ABC', 'Current:Price', '92.8'**

1. Enter the following command to exit the HBase shell.

**quit**

1. Enter the following command to start the Hive command line interface:

**hive**

1. At the Hive prompt, re-enter the following code to query the Hive table again:

**SELECT Stock, CurrentPrice, ClosingPrice,**

**IF(CurrentPrice > ClosingPrice, 'Up',IF (CurrentPrice < ClosingPrice, 'Down', '-'))**

**AS Status**

**FROM StockPrices**

**ORDER BY Stock;**

1. Wait for the MapReduce job to complete, and then view the results. Note that the status for stock ABC is now “Down” (because the Hive table retrieves the latest data from the underlying HBase table each time it is queried, and the current stock price is now lower than the closing price.)
2. Enter the following command to exit the Hive shell.

**quit;**