Cloudera Training for Apache HBase: Hands-On Exercises

General Notes

Cloudera's training courses use a Virtual Machine running the CentOS Linux distribution. This VM has a recent version of CDH installed in Pseudo-Distributed mode. Pseudo-Distributed mode is a method of running Hadoop whereby all five Hadoop daemons run on the same machine. It is essentially a cluster consisting of a single machine. It works just like a larger Hadoop cluster, the only key difference (apart from speed, of course!) being that the block replication factor is set to 1, since there is only a single DataNode available.

Points to note while working in the VM

1. The VM is set to automatically log in as the user `training`. Should you log out at any time, you can log back in as the user `training` with the password `training`.

2. Should you need it, the root password is `training`. You may be prompted for this if, for example, you want to change the keyboard layout. In general, you should not need this password since the `training` user has unlimited sudo privileges.

3. If you restart or suspend the VM for any reason, you may also need to restart HBase manually by issuing the following command:

```
$ ~/scripts/hbase/hbase_restart.sh
```

4. In some command-line steps in the exercises, you will see lines like this:

```
$ hdfs dfs -put shakespeare  \
/user/training/shakespeare
```

   The backslash at the end of the first line signifies that the command is not completed, and continues on the next line. You can enter the code exactly as shown (on two lines), or you can enter it on a single line. If you do the latter, you should *not* type in the backslash.

Hands-On Exercise: Using HDFS

In this exercise you will begin to get acquainted with Hadoop. You will manipulate files in HDFS, the Hadoop Distributed File System.

Hadoop

Hadoop is already installed, configured, and running on your virtual machine. Hadoop is installed in the `/usr/lib/hadoop` directory.

Most of your interaction with the system will be through commands supported by the `bin/hdfs` script. If you start a terminal and run this program with no arguments, it prints a help message. To try this, run the following command:

```
$ hdfs
```

Note: although your command prompt is more verbose, we use '`$`' to indicate the command prompt for brevity.

Step 1: Exploring HDFS

1. Open a terminal window (if one is not already open) by double-clicking the Terminal icon on the desktop.

2. Start the services we'll need for this course by typing the following at the command prompt:

   ```
   $ ~/scripts/hbase/training_setup_hbase.sh
   ```

3. Enter:

   ```
   $ hdfs dfs -ls /
   ```

   This shows you the contents of the root directory in HDFS. There will be multiple entries, one of which is `/user`. Individual users have a "home" directory under

this directory, named after their username – your home directory is `/user/training`.

4. View the contents of the `/user` directory by entering the following:

```
$ hdfs dfs -ls /user
```

You will see your home directory in the directory listing.

5. Enter the following:

```
$ hdfs dfs -ls /user/training
```

There are no files, so the command silently exits. This is different than if you ran `hdfs dfs -ls /foo`, which refers to a directory that doesn't exist and which would display an error message.

Note that the directory structure in HDFS has nothing to do with the directory structure of the local filesystem; they are completely separate namespaces.

Step 2: Uploading Files

Besides browsing the existing filesystem, another important thing you can do with `FsShell` is to upload new data into HDFS.

1. Change directories to the directory containing the sample data we will be using in the course.

```
$ cd ~/training_materials/hbase/data
```

If you perform a 'regular' `ls` command in this directory, you will see a few files, including one named `shakespeare.tar.gz`.

2. Unzip `shakespeare.tar.gz` by running:

```
$ tar zxvf shakespeare.tar.gz
```

This creates a directory named `shakespeare/` containing several files on your local filesystem. If you have used this VM for a previous course, we recommend that you unzip the tar file again to ensure you have the file required for this lab.

3. Change to the shakespeare directory and view the files:

```
$ cd shakespeare
$ ls
```

You will see a few files, one of which is named `glossary`.

4. Insert this file into HDFS:

```
$ hdfs dfs -put glossary /user/training/glossary
```

This copies the local `glossary` file into the remote HDFS directory named `/user/training/`

5. List the contents of your HDFS home directory now:

```
$ hdfs dfs -ls /user/training
```

You should see an entry for the `glossary` file.

6. Now try the same `fs -ls` command but without a path argument:

```
$ hdfs dfs -ls
```

You should see the same results. If you don't pass a directory name to the `-ls` command, it assumes you mean your home directory, i.e. `/user/training`.

7. Enter:

```
$ hdfs dfs -tail glossary
```

This prints the last 1KB of the glossary to your terminal. This command is handy for viewing the output of MapReduce programs. Very often, an individual output file of a MapReduce program is very large, making it inconvenient to view the entire file in the terminal.

Other Commands

There are several other commands associated with the `FsShell` subsystem which let you perform most common filesystem manipulations: `rm`, `rm -r` (recursive `rm`), `mv`, `cp`, `mkdir`, etc.

1. In the terminal window, enter:

```
$ hdfs dfs
```

You see a help message describing all the commands associated with this subsystem.

Bonus Exercise: Advanced `FsShell` Commands

1. In the terminal window, enter:

```
$ hdfs dfs -help
```

This will give a more complete description of each `FsShell` command and the arguments for it.

2. Enter:

```
$ hdfs dfs -cat glossary | tail -n 50
```

This prints the last 50 lines of the glossary to your terminal. This command is handy for viewing the output of MapReduce programs. Very often, an individual output file of a MapReduce program is very large, making it inconvenient to view the entire file in the terminal. For this reason, it's often a good idea to pipe the output of the `fs -cat` command into `head`, `tail`, `more`, or `less`.

This is the end of the Exercise

Hands-On Exercise: HBase Data Import

In this exercise you will import data from MySQL into HBase. Your company was using MySQL and started hitting scale issues as the traffic grew. You have been tasked with migrating the data from MySQL to HBase.

Step 1: Examine MySQL tables

1. Change directories to the directory containing the data import scripts and code.

```
$ cd ~/training_materials/hbase/exercises/DataImport
```

2. Let's first look at the data we will import shortly. The data is in MySQL tables. Issue the following commands to enter the MySQL shell. We will use tables that are in the `movielens` database:

```
$ mysql -u root
mysql> use movielens;
mysql> show tables;
```

3. Issue the following command to count the number of records in the `movie` table. Remember this number for later to verify that all MySQL records were successfully loaded into the HBase `movie` table:

```
mysql> select count(*) from movie;
```

4. Next find out how many records are in the `movierating` table:

```
mysql> select count(*) from movierating;
```

5. Finally, find movie ratings for the user with user ID `4185`:

```
mysql> select * from movierating where userid = 4185;
```

6. Exit the MySQL shell:

```
mysql> exit;
```

Step 2: Create HBase tables

1. Next create the HBase tables that we'll load with data from the MySQL tables we looked at above. First, create a new HBase table called `movie` by issuing:

```
$ echo "create 'movie', {NAME => 'info', VERSIONS => 3},
{NAME => 'media', VERSIONS => 3}" | hbase shell
```

   This creates a new table in HBase named `movie` that does not contain any rows. We will use another command to move the rows from MySQL into HBase.

2. Create another new HBase table called `user` by issuing:

```
$ echo "create 'user', {NAME => 'info', VERSIONS => 3},
{NAME => 'ratings', VERSIONS => 3}" | hbase shell
```

   This creates another new table in HBase named `user` that does not contain any rows. We will use a different command to move the rows from MySQL into HBase.

Step 3: Load MySQL data into HBase tables

1. Import the movie data from MySQL by running:

```
$ ./moviesimport.sh
```

   This will compile and run a Java program that imports the data with the HBase API. The program will connect to MySQL, process the data, and use the HBase API to put the data in HBase.

2. Import the user information from MySQL by running:

```
$ ./usersimport.sh
```

This will compile and run a Java program that imports the data with the HBase API. The program will connect to MySQL, process the data, and use the HBase API to put the data in HBase.

3. Import the users' movie ratings from MySQL by running:

```
$ ./userratingsimport.sh
```

This will compile and run a Java program that imports the data with the HBase API. The program will connect to MySQL, process the data, and use the HBase API to put the data in HBase.

This is the end of the Exercise

Hands-On Exercise: Using the HBase Shell

In this exercise you will begin to get acquainted with the HBase Shell. You will perform basic table functions including creating, deleting, and altering a table. Sample solutions are available in the next section.

Step 1: Experiment with creating and deleting tables

1. Invoke the HBase shell
2. View the help menu.
3. List all tables.
4. Create a table with the following characteristics:
   Table name: hbase_tbl_1
   Column Family: column_fam_1
5. List all tables again to verify table `hbase_tbl_1` was successfully created.
6. Describe the table you just created.
7. Delete the table you just created.
8. List all tables again to verify table `hbase_tbl_1` was successfully deleted.

Step 2: Work with column families

1. Count the number of rows in the HBase `movie` table that we created in the previous exercise:

```
hbase> count 'movie'
```

2. Verify that the number of rows counted matches the number of rows in the MySQL `movie` table noted in the previous exercise.

3. Update the `movie` table to add a `subtitles` column family.

4. Describe the `movie` table to verify `subtitles` column family has been added.

5. Remove the `subtitles` column family from the `movie` table.

6. Describe the `movie` table to verify `subtitles` column family has been removed.

7. Modify the `media` column family in the `movie` table to keep four versions.

8. Describe the `movie` table to verify the changes.

9. Modify the `media` column family in the `movie` table to keep only one version.

10. Describe the `movie` table to verify the changes.

11. Using another terminal, view the HDFS directory structure:

```
$ hdfs dfs -ls /hbase/data/default
$ hdfs dfs -ls /hbase/data/default/movie/*
```

You will see that both the user and movie tables are subdirectories in `/hbase/data/default`. Also notice that the `info` and `media` column families are shown – they are subdirectories.

This is the end of the Exercise

Hands-On Exercise Solution: Using the HBase Shell

Step 1: Experiment with creating and deleting tables

1. Start the HBase shell.

```
$ hbase shell
```

2. View the help menu.

```
hbase> help
```

3. List all tables.

```
hbase> list
```

4. Create a table with the following characteristics:
   Table name: hbase_tbl_1
   Column Family: column_fam_1

```
hbase> create 'hbase_tbl_1', 'column_fam_1'
```

5. List all tables again to verify table `hbase_tbl_1` was successfully created.

```
hbase> list
```

6. Describe the table you just created.

```
hbase> describe 'hbase_tbl_1'
```

7. Delete the table you just created.

```
hbase> disable 'hbase_tbl_1'
```

```
hbase> drop 'hbase_tbl_1'
```

8. List all tables again to verify table `hbase_tbl_1` was successfully deleted.

```
hbase> list
```

Step 2: Work with column families

1. Count the number of rows in the HBase `movie` table that we created in the previous exercise:

```
hbase> count 'movie'
```

2. Verify that the number of rows counted matches the number of rows in the MySQL `movie` table noted in the previous exercise.

3. Update the `movie` table to add a `subtitles` column family.

```
hbase> alter 'movie', NAME => 'subtitles'
```

4. Describe the `movie` table to verify `subtitles` column family has been added.

```
hbase> describe 'movie'
```

5. Remove the `subtitles` column family from the `movie` table.

```
hbase> alter 'movie', NAME => 'subtitles', METHOD =>
'delete'
```

6. Describe the `movie` table to verify `subtitles` column family has been removed.

```
hbase> describe 'movie'
```

7. Modify the `media` column family in the `movie` table to keep four versions.

```
hbase> alter 'movie', NAME => 'media', VERSIONS => 4
```

8. Describe the `movie` table to verify the changes.

```
hbase> describe 'movie'
```

9. Modify the `media` column family in the `movie` table to keep only one version.

```
hbase> alter 'movie', NAME => 'media', VERSIONS => 1
```

10. Describe the `movie` table to verify the changes.

```
hbase> describe 'movie'
```

11. Using another terminal, view the HDFS directory structure:

```
$ hdfs dfs -ls /hbase/data/default
$ hdfs dfs -ls /hbase/data/default/movie/*
```

12. You will see that both the `user` and `movie` tables are subdirectories in `/hbase/data/`default. Also notice that the `info and media` column families are shown – they are subdirectories of the `movie` subdirectory.

This is the end of the Exercise

Hands-On Exercise: Data Access in the HBase Shell

In this exercise you will use the HBase Shell to put and get data.

Step 1: Add data to HBase tables

1. Invoke the HBase shell:

```
$ hbase shell
```

2. Describe the `user` table to remind yourself of the column family names.

3. Get the row with row key `10000` from the `user` table. The row should not exist.

4. Add a row with the following characteristics:

   Table name: 'user'
   Row key: '10000'
   Column Family: 'info'
   Column Descriptor: 'age' and value '1'
   Column Descriptor: 'gender' and value 'F'
   Column Descriptor: 'occupationname' and value 'Toddling'
   Column Descriptor: 'zip' and value '90210'

5. Get the row with row key `10000` from the `user` table. The row should exist now.

6. Do another put with the following characteristics:

   Table name: ‘user’
   Row key: ‘10000’
   Column Family: ‘info’
   Column Descriptor: ‘age’ and value ‘2’

7. Do a third put with the following characteristics:

Table name: 'user'
Row key: '10000'
Column Family: 'info'
Column Descriptor: 'age' and value '3'

8. Get the row with row key `10000` from the `user` table. Note that the value for the `age` is 3.

9. Get all of the previous versions of the `age` column with the `10000` row key from the `user` table.

10. View the entire table, but only look at the `age` column:

```
scan 'user', {COLUMNS => 'info:age'}
```

Step 2: Delete columns and rows from HBase tables

1. Delete the `age` column descriptor from the `user` table with the `10000` row key.

2. Verify that the `age` column descriptor has been removed.

3. Delete the entire row from the `user` table with the `10000` row key.

4. Verify that the row with row key 10000 has been removed from table `user`.

Step 3: Explore HBase directories in HDFS

1. Using another terminal, view the HBase directory structure in HDFS:

```
$ hdfs dfs -ls /hbase/data/default
```

This directory holds all of the tables for HBase and some other files used by HBase.

2. View the directory structure of a table in HDFS:

```
$ hdfs dfs -ls /hbase/data/default/user/*
```

This directory shows how a table is broken in to regions.

This is the end of the Exercise

Hands-On Exercise Solution: Data Access in the HBase Shell

Step 1: Add data to HBase tables

1. Invoke the HBase shell:

```
$ hbase shell
```

2. Describe the `user` table to remind yourself of the Column Family names.

```
hbase> describe 'user'
```

3. Get the row with row key `10000` from the `user` table. The row should not exist.

```
hbase> get 'user', '10000'
```

4. Add a row with the following characteristics:

   Table name: 'user'
   Row key: '10000'
   Column Family: 'info'
   Column Descriptor: 'age' and value '1'
   Column Descriptor: 'gender' and value 'F'
   Column Descriptor: 'occupationname' and value 'Toddling'
   Column Descriptor: 'zip' and value '90210'

```
hbase> put 'user', '10000', \
'info:age', '1'
hbase> put 'user', '10000', \
'info:gender', 'F'
hbase> put 'user', '10000', \
'info:occupationname', 'Toddling'
hbase> put 'user', '10000', \
'info:zip', '90210'
```

5. Get the row with row key `10000` from the `user` table. The row should exist now.

```
hbase> get 'user', '10000'


Result:
COLUMN                CELL
 info:age             timestamp=1425409330107, value=1
 info:gender          timestamp=1425409340833, value=F
 info:occupationname  timestamp=1425409354472, value=Toddling
 info:zip             timestamp=1425409362046, value=90210
4 row(s) in 0.0210 seconds
```

6. Do another put with the following characteristics:

   Table name: 'user'
   Row key: '10000'
   Column Family: 'info'
   Column Descriptor: 'age' and value '2'

```
hbase> put 'user', '10000', 'info:age', '2'
```

7. Do a third put with the following characteristics:

Table name: ‘user’
Row key: ‘10000’
Column Family: ‘info’
Column Descriptor: ‘age’ and value ‘3’

```
hbase> put 'user', '10000', 'info:age', '3'
```

8. Get the row with row key `10000` from the `user` table. Note that the value for the `age` is 3.

```
hbase> get 'user', '10000'

Result:
COLUMN                CELL
 info:age             timestamp=1425409531363, value=3
 info:gender          timestamp=1425409340833, value=F
 info:occupationname  timestamp=1425409354472,
value=Toddling
 info:zip             timestamp=1425409362046,
value=90210
4 row(s) in 0.0250 seconds
```

9. Get all of the previous versions of the `age` column with the `10000` row key from the `user` table.

```
hbase> get 'user', '10000', \
{COLUMN => 'info:age', VERSIONS => 3}

Result:
COLUMN               CELL
 info:age            timestamp=1425409531363, value=3
 info:age            timestamp=1425409513005, value=2
 info:age            timestamp=1425409330107, value=1
3 row(s) in 0.0280 seconds
```

10. View the entire table, but only look at the `age` column:

```
scan 'user', {COLUMNS => 'info:age'}
```

Step 2: Delete columns and rows from HBase tables

1. Delete the `age` Column Descriptor from the `user` table with the `10000` row key.

```
hbase> delete 'user', '10000', 'info:age'
```

2. Verify that the `age` column has been removed:

```
hbase> get 'user', '10000'


Result:
COLUMN                CELL
 info:gender          timestamp=1425409340833, value=F
 info:occupationname  timestamp=1425409354472, value=Toddling
 info:zip             timestamp=1425409362046, value=90210
3 row(s) in 0.0440 seconds
```

3. Delete the entire row from the `user` table with the `10000` row key.

```
hbase> deleteall 'user', '10000'
```

4. Verify that the row with row key 10000 has been removed from table `user`.

```
hbase> get 'user', '10000'


Result:
COLUMN                CELL
0 row(s) in 0.0090 seconds
```

Step 3: Explore HBase directories in HDFS

1. Using another terminal, view the HBase directory structure in HDFS:

```
$ hdfs dfs -ls /hbase/data/default


Results:
Found 2 items
drwxrwxrwx - hbase supergroup  0 2015-03-03 09:15 /hbase/data/default/movie
drwxrwxrwx - hbase supergroup  0 2015-03-03 09:15 /hbase/data/default/user
```

This directory holds all of the tables for HBase and some other files used by HBase.

2. View the directory structure of a table in HDFS:

```
$ hdfs dfs -ls /hbase/data/default/user/*
```

Results:

```
Found 1 items
-rw-rw-rw-   1 hbase supergroup        532 2015-03-03 09:15
/hbase/data/default/user/.tabledesc/.tableinfo.0000000001
Found 4 items
-rw-rw-rw-   1 hbase supergroup         37 2015-03-03 09:15
/hbase/data/default/user/05e80d00eaba09746e538104ef7d5e2d/.regioninfo
drwxrwxrwx   - hbase supergroup          0 2015-03-03 10:20
/hbase/data/default/user/05e80d00eaba09746e538104ef7d5e2d/.tmp
drwxrwxrwx   - hbase supergroup          0 2015-03-03 09:20
/hbase/data/default/user/05e80d00eaba09746e538104ef7d5e2d/info
drwxrwxrwx   - hbase supergroup          0 2015-03-03 10:20
/hbase/data/default/user/05e80d00eaba09746e538104ef7d5e2d/ratings
```

This is the end of the Exercise