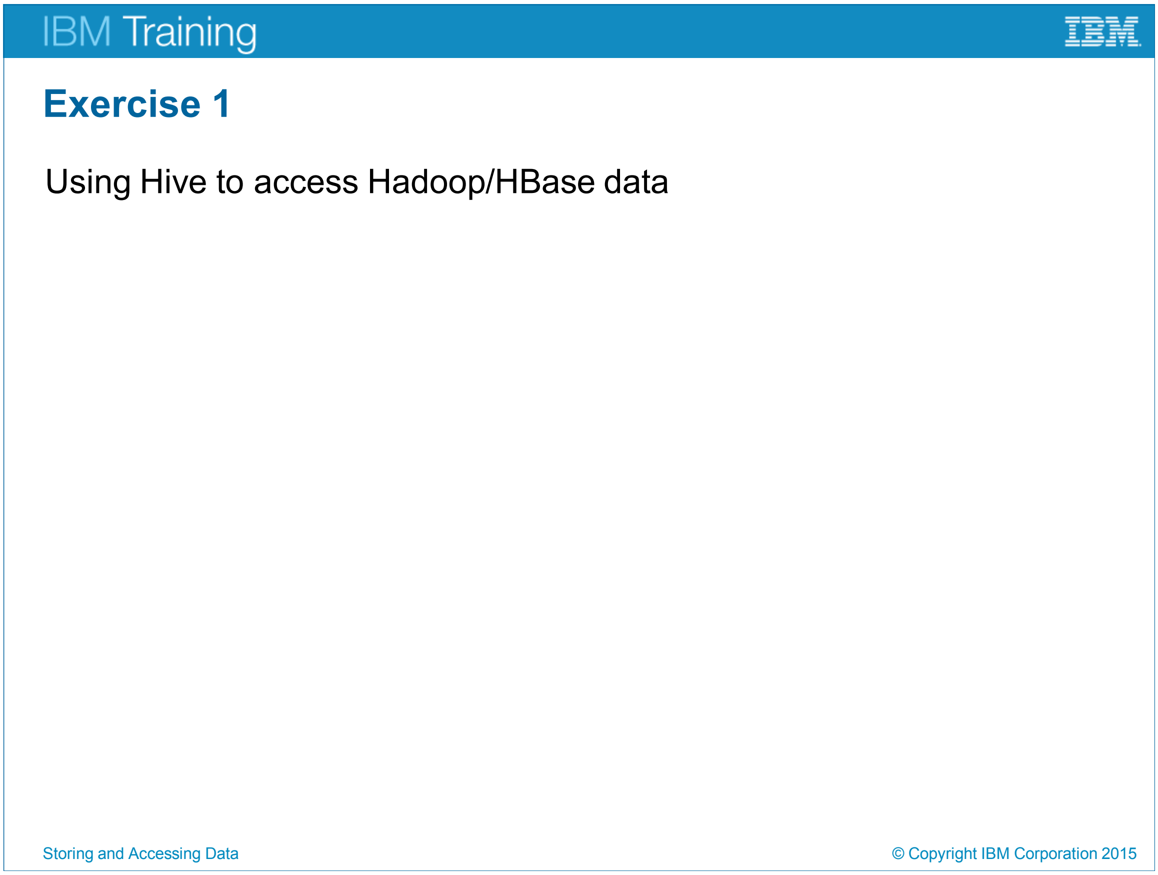


IBM Training

Exercise 1

Using Hive to access Hadoop/HBase data

Storing and Accessing Data

© Copyright IBM Corporation 2015

Exercise 1: Using Hive to access Hadoop/HBase data

Exercise 1:
Using Hive to access Hadoop/HBase data

Purpose:
This exercise is intended to provide you with experience in accessing Hadoop/HBase data using a command line interface (CLI).

Task 1. Storing and accessing HBase data.

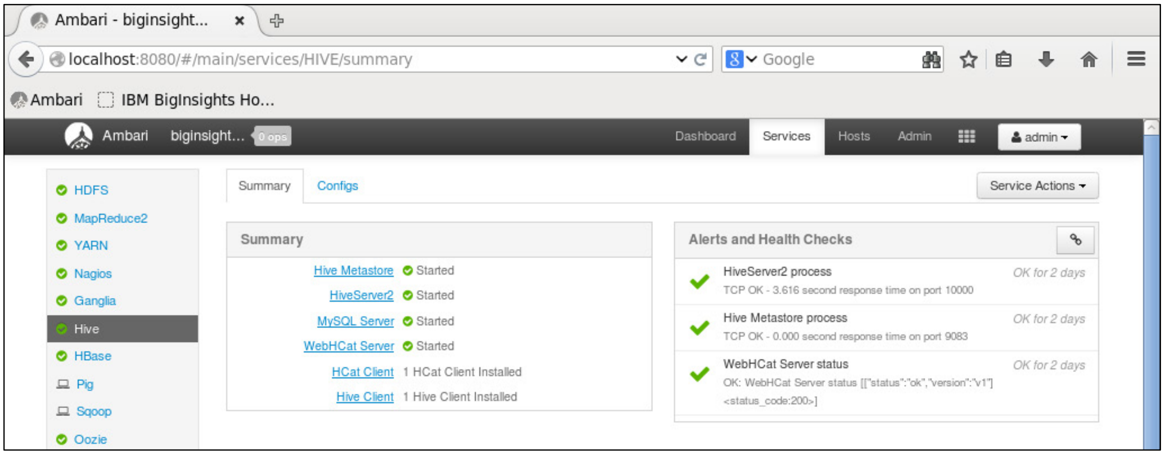
The major references for the HBase can be found on the Apache.org and Apache Wiki websites:

- http://hbase.apache.org
- http://wiki.apache.org/hadoop/Hbase

Big Data University has a free courses on HBase at:

- http://bigdatauniversity.com/bdu-wp/bdu-course/using-hbase-for-real-time-access-to-your-big-data-version-2

1. Connect to and login to your lab environment with user **biadmin** and password **biadmin** credentials.
2. Launch **Firefox**, and then if necessary, navigate to the **Ambari** login page, **http://localhost:8080**, logging in as **admin/admin**.
3. Verify that Hive is running by clicking on **Hive** in the left panel:



Hive Metastore	Started
HiveServer2	Started
MySQL Server	Started
WebHCat Server	Started
HCat Client	1 HCat Client Installed
Hive Client	1 Hive Client Installed

✓ HiveServer2 process — OK for 2 days
TCP OK - 3.616 second response time on port 10000

✓ Hive Metastore process — OK for 2 days
TCP OK - 0.000 second response time on port 9083

✓ WebHCat Server status — OK for 2 days
OK: WebHCat Server status [["status":"ok","version":"v1"] <status_code:200>]

If Hive is not running, you will have to start it using the central panel.

When running, minimize the Ambari Web Console browser.

4. In a new terminal window, type `cd` to change to your home directory.

5. To start the **HBase CLI** shell, type the following command:

```
/usr/bin/hbase shell
```

```
[biadmin@ibmclass ~]$ /usr/bin/hbase shell
2015-06-07 11:57:36,247 INFO  [main] Configuration.deprecation: hadoop.native.lib is
deprecated. Instead, use io.native.lib.available
HBase Shell; enter 'help<RETURN>' for list of supported commands.
Type "exit<RETURN>" to leave the HBase Shell
Version 0.98.8_IBM_4-hadoop2, rUnknown, Fri Mar 27 21:53:57 PDT 2015

hbase(main):001:0>
```

The last line here is the prompt for the Hbase CLI Client. Note that the interactions are numbered (001) for each CLI session.

6. To view the online CLI help manual, type `help` at the prompt and then press **Enter**:

```
hbase(main):001:0> help
HBase Shell, version 0.98.8_IBM_4-hadoop2, rUnknown, Fri Mar 27 21:53:57 PDT 2015
Type 'help "COMMAND"', (e.g. 'help "get"' -- the quotes are necessary) for help on a
specific command.
Commands are grouped. Type 'help "COMMAND_GROUP"', (e.g. 'help "general"') for help on a
command group.

COMMAND GROUPS:
  Group name: general
  Commands: status, table_help, version, whoami

  Group name: ddl
  Commands: alter, alter_async, alter_status, create, describe, disable, disable_all,
drop, drop_all, enable, enable_all, exists, get_table, is_disabled, is_enabled, list,
show_filters

  Group name: namespace
  Commands: alter_namespace, create_namespace, describe_namespace, drop_namespace,
list_namespace, list_namespace_tables

  Group name: dml
  Commands: append, count, delete, deleteall, get, get_counter, incr, put, scan, truncate,
truncate_preserve

  Group name: tools
  Commands: assign, balance_switch, balancer, catalogjanitor_enabled, catalogjanitor_run,
catalogjanitor_switch, close_region, compact, compact_rs, flush, hlog_roll, major_compact,
merge_region, move, split, trace, unassign, zk_dump

  Group name: replication
  Commands: add_peer, disable_peer, enable_peer, list_peers, list_replicated_tables,
remove_peer, set_peer_tableCFs, show_peer_tableCFs

  Group name: snapshots
  Commands: clone_snapshot, delete_snapshot, list_snapshots, rename_snapshot,
restore_snapshot, snapshot

  Group name: security
  Commands: grant, revoke, user_permission

  Group name: visibility labels
  Commands: add_labels, clear_auths, get_auths, set_auths, set_visibility

SHELL USAGE:
Quote all names in HBase Shell such as table and column names. Commas delimit
```

```
command parameters. Type <RETURN> after entering a command to run it.
Dictionaries of configuration used in the creation and alteration of tables are
Ruby Hashes. They look like this:

  {'key1' => 'value1', 'key2' => 'value2', ...}

and are opened and closed with curley-braces. Key/values are delimited by the
'=>' character combination. Usually keys are predefined constants such as
NAME, VERSIONS, COMPRESSION, etc. Constants do not need to be quoted. Type
'Object.constants' to see a (messy) list of all constants in the environment.

If you are using binary keys or values and need to enter them in the shell, use
double-quote'd hexadecimal representation. For example:

  hbase> get 't1', "key\x03\x3f\xcd"
  hbase> get 't1', "key\003\023\011"
  hbase> put 't1', "test\xef\xff", 'f1:', "\x01\x33\x40"

The HBase shell is the (J)Ruby IRB with the above HBase-specific commands added.
For more on the HBase Shell, see http://hbase.apache.org/book.html
hbase(main):002:0>
```

Take a minute to look through the Help to see what is available to you. Note that in the commands, the names of the tables, rows, column families are all in quotes. You will need to make sure that when you are referring to specific tables, rows, column families, that they are enclosed in quotes.

Practical notes:

- Command (such as create) must be lowercase
- Table name (such as t1) must be quoted, …
- In interactive mode, you do not need a semicolon as statement terminator or separator (unlike standard SQL)

7. Create an Hbase table using the **create** command:

    `create 't1', 'cf1', 'cf2', 'cf3'`

   to create a table t1 with three column families (cf1, cf2, cf3).

```
hbase(main):002:0> create 't1', 'cf1', 'cf2', 'cf3'
SLF4J: Class path contains multiple SLF4J bindings.
SLF4J: Found binding in [jar:file:/usr/iop/4.0.0.0/hadoop/lib/slf4j-log4j12-
1.7.5.jar!/org/slf4j/impl/StaticLoggerBinder.class]
SLF4J: Found binding in [jar:file:/usr/iop/4.0.0.0/zookeeper/lib/slf4j-log4j12-
1.6.1.jar!/org/slf4j/impl/StaticLoggerBinder.class]
SLF4J: See http://www.slf4j.org/codes.html#multiple_bindings for an explanation.
0 row(s) in 5.9840 seconds

=> Hbase::Table - t1
hbase(main):003:0>
```

The table *t1* has been created.

Note that these single quotes are inch-type quotes (') and not Microsoft smart-quotes (‘’). Take care anytime, here and elsewhere, if you cut-and-paste code that Microsoft or other products have not changed the original code available to use by converting to "smart-quotes".

Other notes:

- The create command only requires the name of the table and one or more column families. Columns can be added dynamically to the table. Also, each row can have a different set of columns (within each column family). However, the table may not be mappable to SQL in such cases.
- Our column family names have been deliberately kept short. This is a best practice: keep your column family names short. For example, instead of 'col_fam1' use 'cf1'. HBase stores the entire names across all of their nodes where the data resides. If you use a long name, it will get repeated across all of the nodes increase the total usage. You want to avoid this by using as short of a name as possible.
- The table name does not need to be short. The name t1 here is short, and cryptic, merely to save your typing convenience. In reality you should use more expressive names as that is better documentation of your data model.

8. Type **`describe 't1'`** to verify your table creation.

```
hbase(main):001:0> describe 't1'
SLF4J: Class path contains multiple SLF4J bindings.
SLF4J: Found binding in [jar:file:/usr/iop/4.0.0.0/hadoop/lib/slf4j-log4j12-
1.7.5.jar!/org/slf4j/impl/StaticLoggerBinder.class]
SLF4J: Found binding in [jar:file:/usr/iop/4.0.0.0/zookeeper/lib/slf4j-log4j12-
1.6.1.jar!/org/slf4j/impl/StaticLoggerBinder.class]
SLF4J: See http://www.slf4j.org/codes.html#multiple_bindings for an explanation.
Table t1 is ENABLED
t1
COLUMN FAMILIES DESCRIPTION
{NAME => 'cf1', DATA_BLOCK_ENCODING => 'NONE', BLOOMFILTER => 'ROW', REPLICATION_SCOPE =>
'0', VERSIONS => '1', COMPRESSION => 'NONE', M
IN_VERSIONS => '0', TTL => 'FOREVER', KEEP_DELETED_CELLS => 'FALSE', BLOCKSIZE => '65536',
IN_MEMORY => 'false', BLOCKCACHE => 'true'}
{NAME => 'cf2', DATA_BLOCK_ENCODING => 'NONE', BLOOMFILTER => 'ROW', REPLICATION_SCOPE =>
'0', VERSIONS => '1', COMPRESSION => 'NONE', M
IN_VERSIONS => '0', TTL => 'FOREVER', KEEP_DELETED_CELLS => 'FALSE', BLOCKSIZE => '65536',
IN_MEMORY => 'false', BLOCKCACHE => 'true'}
{NAME => 'cf3', DATA_BLOCK_ENCODING => 'NONE', BLOOMFILTER => 'ROW', REPLICATION_SCOPE =>
'0', VERSIONS => '1', COMPRESSION => 'NONE', M
IN_VERSIONS => '0', TTL => 'FOREVER', KEEP_DELETED_CELLS => 'FALSE', BLOCKSIZE => '65536',
IN_MEMORY => 'false', BLOCKCACHE => 'true'}
3 row(s) in 1.4450 seconds

hbase(main):002:0>
```

Note the following metadata features in your describe response:

- COMPRESSION
- IN_MEMORY
- VERSIONS

You will be changing some of these values shortly with an alter statement:

The next question is: just where is the table t1 stored? It is stored in HDFS, but where in particular?

9. Open another terminal window (so that you can continue later in the first terminal window) and execute the following command:

    ```
    hadoop fs -ls -R / 2>/dev/null | grep t1
    ```

    where this command lists all files (recursive, **-R**) and passes the results to the Linux command grep. There would be errors, but these are discarded (2>/dev/null).

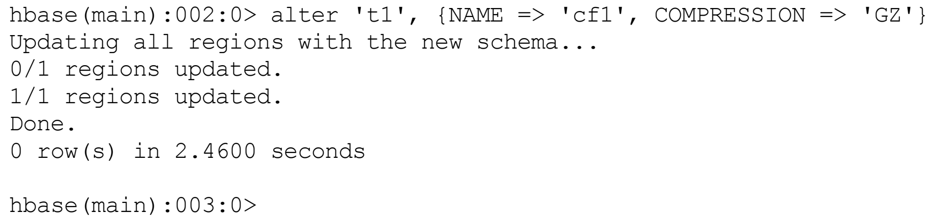
    ```
    [biadmin@ibmclass Desktop]$ hadoop fs -ls -R / 2>/dev/null | grep t1
    drwxr-xr-x   - hbase     hdfs            0 2015-06-07 12:17
    /apps/hbase/data/data/default/t1
    drwxr-xr-x   - hbase     hdfs            0 2015-06-07 12:17
    /apps/hbase/data/data/default/t1/.tabledesc
    -rw-r--r--   3 hbase     hdfs          769 2015-06-07 12:17
    /apps/hbase/data/data/default/t1/.tabledesc/.tableinfo.0000000001
    drwxr-xr-x   - hbase     hdfs            0 2015-06-07 12:17
    /apps/hbase/data/data/default/t1/.tmp
    drwxr-xr-x   - hbase     hdfs            0 2015-06-07 12:17
    /apps/hbase/data/data/default/t1/8a45456f26ee4569360c6af03e893ed6
    -rw-r--r--   3 hbase     hdfs           35 2015-06-07 12:17
    /apps/hbase/data/data/default/t1/8a45456f26ee4569360c6af03e893ed6/.regioninfo
    drwxr-xr-x   - hbase     hdfs            0 2015-06-07 12:17
    /apps/hbase/data/data/default/t1/8a45456f26ee4569360c6af03e893ed6/cf1
    drwxr-xr-x   - hbase     hdfs            0 2015-06-07 12:17
    /apps/hbase/data/data/default/t1/8a45456f26ee4569360c6af03e893ed6/cf2
    drwxr-xr-x   - hbase     hdfs            0 2015-06-07 12:17
    /apps/hbase/data/data/default/t1/8a45456f26ee4569360c6af03e893ed6/cf3
    [biadmin@ibmclass Desktop]$
    ```

    Note that directories are created. Files will be put into those directories as records are stored into the table t1.

    For HBase packaged with BigInsights, only gzip compression can be used out of the box. For this, you will use the alter command.

10. In the first terminal window, specify compression for a column family in the table using the following statement:

    `alter 't1', {NAME => 'cf1', COMPRESSION => 'GZ'}`



```
hbase(main):002:0> alter 't1', {NAME => 'cf1', COMPRESSION => 'GZ'}
Updating all regions with the new schema...
0/1 regions updated.
1/1 regions updated.
Done.
0 row(s) in 2.4600 seconds

hbase(main):003:0>
```

The other compression algorithms that are supported but would need extra configuration are SNAPPY and LZO. Note that GZIP is slow but also has the most efficient compression option.

Sometimes you may find that you need to disable the table prior to executing the alter statement:

```
disable 't1'
alter 't1', {NAME
```

You will now make additional changes to the metadata for the table.

11. Specify the IN_MEMORY option for a column family that will be queried frequently. This does not ensure the data will be in memory always. It only gives priority for the corresponding data to stay in the cache longer.

    ```
    alter 't1', {NAME => 'cf1', IN_MEMORY => 'true'}
    ```

12. Specify the required number of versions for a column. By default, HBase stores 1 version of the value, but you can set to have more than 1 versions stored. Enter the following in one continuous line:

    ```
    alter 't1', {NAME => 'cf1', VERSIONS => 3},
    {NAME => 'col_fam2', VERSIONS => 2}
    ```

13. Run the describe statement again, and verify that these changes were made to the table and the column families.

    ```
    describe 't1'
    ```

14. Insert dates into the table using the put command. Each row could have different set of columns. The below set of put commands inserts two rows with a different set of column names. Go ahead and enter each of these commands (singly or as a group) into the HBase CLI shell, or insert something similar of your choice.

```
put 't1', 'row1', 'cf1:c11', 'r1v11'
put 't1', 'row1', 'cf1:c12', 'r1v12'
put 't1', 'row1', 'cf2:c21', 'r1v21'
put 't1', 'row1', 'cf3:c31', 'r1v31'
put 't1', 'row2', 'cf1:d11', 'r2v11'
put 't1', 'row2', 'cf1:d12', 'r2v12'
put 't1', 'row2', 'cf2:d21', 'r2v21'
```

```
hbase(main):010:0> put 't1', 'row1', 'cf1:c11', 'r1v11'
put 't1', 'row1', 'cf1:c12', 'r1v12'
put 't1', 'row1', 'cf2:c21', 'r1v21'
put 't1', 'row1', 'cf3:c31', 'r1v31'
put 't1', 'row2', 'cf1:d11', 'r2v11'
put 't1', 'row2', 'cf1:d12', 'r2v12'
put 't1', 'row2', 'cf2:d21', 'r2v21'
0 row(s) in 0.3680 seconds

0 row(s) in 0.0410 seconds

0 row(s) in 0.0590 seconds

0 row(s) in 0.0890 seconds

0 row(s) in 0.0520 seconds

0 row(s) in 0.0420 seconds

0 row(s) in 0.0550 seconds

hbase(main):016:0>
```

15. To view the data, you may use the **get** command to retrieve an individual row, or the **scan** command to retrieve multiple rows:

```
get 't1', 'row1'
```

```
hbase(main):021:0> get 't1', 'row1'
COLUMN                                CELL
 cf1:c11                              timestamp=1433702916069, value=r1v11
 cf1:c12                              timestamp=1433702916309, value=r1v12
 cf2:c21                              timestamp=1433702916379, value=r1v21
 cf3:c31                              timestamp=1433702916476, value=r1v31
4 row(s) in 0.0570 seconds
```

Curiosity point: All data is versioned either using an integer timestamp (seconds since the epoch, 1 Jan 1970 UCT/GMT), or another integer of your choice.

16. If you run the **scan** command, you will see a per-row listing of values:

    `scan 't1'`

```
hbase(main):022:0> scan 't1'
ROW                                  COLUMN+CELL
   row1                                column=cf1:c11, timestamp=1433702916069, value=r1v11
   row1                                column=cf1:c12, timestamp=1433702916309, value=r1v12
   row1                                column=cf2:c21, timestamp=1433702916379, value=r1v21
   row1                                column=cf3:c31, timestamp=1433702916476, value=r1v31
   row2                                column=cf1:d11, timestamp=1433702916539, value=r2v11
   row2                                column=cf1:d12, timestamp=1433702916595, value=r2v12
   row2                                column=cf2:d21, timestamp=1433702916661, value=r2v21
2 row(s) in 0.1230 seconds

hbase(main):023:0>
```

Notes:

- The above scan results show that HBase tables do not require a set schema. This is good for some applications that need to store arbitrary data. To put this in other words, HBase does not store null values. If a value for a column is null (e.g. values for d11, d12, d21 are null for row1), it is not stored. This is one aspect that makes HBase work well with sparse data.
- In addition to the actual column value (*r1v11*), each result row has the row key value (*row1*), column family name (*col_fam1*), column qualifier/column (*c11*) and timestamp. These pieces of information are also stored physically for each value. Having a large number of columns with values for all rows (in other words, dense data) would mean this information gets repeated. Also, larger row key values, longer column family and column names would increase the storage space used by a table. For example use r1 instead of row1.

Good business practices:

- Try to use smaller row key values, column family and qualifier names.
- Try to use fewer columns if you have dense data

Task 2. Storing and accessing HBase data.

The major references for the Hive can be found on the Apache.org and Apache Wiki websites:

- https://hive.apache.org
- https://cwiki.apache.org/confluence/display/Hive/Tutorial

Big Data University (BDU) has a free courses on Hive:

- http://bigdatauniversity.com/bdu-wp/bdu-course/accessing-hadoop-data-using-hive-version-2

You will not have time in this unit to do a full exercise on Hive, but you will start the Hive CLI client to learn where to find it.

1. In a new terminal window, type **`cd`** to change to the home directory.
2. To start the Hive client, type **`hive`**.

```
[biadmin@ibmclass ~]$ hive
15/06/07 14:57:18 WARN conf.HiveConf: HiveConf of name hive.optimize.mapjoin.mapreduce
does not exist
15/06/07 14:57:18 WARN conf.HiveConf: HiveConf of name hive.heapsize does not exist
15/06/07 14:57:18 WARN conf.HiveConf: HiveConf of name
hive.auto.convert.sortmerge.join.noconditionaltask does not exist

Logging initialized using configuration in file:/etc/hive/conf/hive-log4j.properties
hive>
```

Some configuration is needed for Hive. With the appropriate configuration and setup, the HBase table that you created can be accessed with Hive and HiveQL.

It is recommended that you take the BDU course and/or continue your learning with one of the many tutorials available on the internet.

3. Close all open windows.

Results:
You accessed Hadoop/HBase data using a command line interface (CLI).