

# Lab: Exploring HBase 1

**Objective:** Use the HBase Shell to populate and explore the `hbase:meta` table.

**HDFS paths:** `/hbase/data/default` `/hbase/data/default/tags`  
`/hbase/data/default/movie`

**HBase tables:** `movies`, `tags`, `hbase:meta`, `[some name]`

**Data Source:** [grouplens.org/datasets](https://grouplens.org/datasets)

In this exercise you will use the HBase Shell to explore the `hbase:meta` table.

---

```

<h4>1. The HBase Shell</h4>
```

Invoke the HBase shell and print the help menu:

```
hbase shell
```

Get the status of the HBase cluster:

```
hbase(main):000:0> status 'simple'
```

```
1 live servers
```

```
localhost:60020 1425572833316 requestsPerSecond=0.0, numberOfOnlineRegions=4,usedHeapMB=72, maxHeapMB=503, numberOfStores=6, numberofStorefiles=8, storefileUncompressedSizeMB=34, storefileSizeMB=34, compressionRatio=1.0000, memstoreSizeMB=0, storefileIndexSizeMB=0, readRequestsCount=82856, writeRequestsCount=37055, rootIndexSizeKB=43, totalStaticIndexSizeKB=20, totalStaticBloomSizeKB=32, totalCompactingKVs=33915, currentCompactedKVs=33915, compactionProgressPct=1.0, coprocessors=[]
```

```
0 dead servers
```

```
Aggregate load: 0, regions: 4
```

This command shows the status of all nodes in the cluster. It breaks the information down into live and dead servers.

The live servers show information about the load and number of regions each node is handling. It will display

details about the system status like a number of servers present in the cluster, active server count, and average load value.

You can also pass parameters depending on how detailed status you want.

The parameters can be 'summary', 'simple', or 'detailed'. The default is 'summary'.

You can also get a detailed status by adding **detailed** to the status

command:

```
hbase(main):000:0> status 'detailed'
version 0.96.0.2.0.6.0-76-hadoop2
0   regionsInTransition master coprocessors: [ ]
1   live servers
    sandbox.hortonworks.com:60020 1390046418833 requestsPerSecond=0.0, numberOfOnlineRegions=4,
    usedHeapMB=102, maxHeapMB=1004, numberOfStores=4, numberOfStorefiles=4, storefileUncompressedSizeMB=14, storefileSizeMB=14,
    compressionRatio=1.0000, memstoreSizeMB=0, storefileIndexSizeMB=0, readRequestsCount=1231, writeRequestsCount=46,
    rootIndexSizeKB=16,
    totalStaticIndexSizeKB=8, totalStaticBloomSizeKB=64, totalCompactingKVs=37,
    ...
```

Now do more status check for summary:

```
hbase(main):000:0> status 'summary'
```

Determine what user you are connected as. Use the whoami command:

```
hbase> whoami
root (auth:SIMPLE)
```

```

<h4>2. Getting Help</h4>
```

Run the help command and view the output:

```
hbase(main):027:0> help
```

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.

```

<h4>3. HBase Meta Table</h4>
```

Now scan the `hbase:meta` It will give information about the tables that HBase is serving (your results may vary):

```
hbase> scan 'hbase:meta'
```

ROW	COLUMN+CELL
-----	-------------

hbase:namespace,	... column=info:regioninfo, timestamp=1425389660
movie,	...column=info:regioninfo, timestamp=14254029136
movie, ...	column=info:seqnumDuringOpen, timestamp=1425572846014,value=\x00\x00\x00\x00 ...
movie, ...	column=info:server, timestamp=1425572846014,value=localhost:60020,
movie, ...	column=info:serverstartcode, timestamp=1425572846014,value=1425572833316
user, ...	column=info:regioninfo, timestamp=142540293339
...	

The `hbase:meta` table contains information about the node that is serving the table. It also keeps track of the start and end keys for the region. Clients use this information to know which node or RegionServer to contact to access a row.

```

<h4>4. Create a Table</h4>
```

Now let's create a new table (whatever name you wish):

```
hbase> create [some name], 'cf1', 'cf2', {SPLITS => [ '

```

```
A', 'M', 'Z']}]}
```

When creating a new table in HBase, you can split the table into regions as the starting point. The splits passed in during the create will serve as the initial regions for the table.

Now again get the status of the HBase cluster:

```
hbase> status 'simple'
```

```
1 live servers
```

```
localhost:60020 1425572833316 requestsPerSecond=0.0, number0f0nlineRegions=8,
    usedHeapMB=71, maxHeapMB=503, number0fStores=14, number0fStorefiles=8, storefileUncompressedSizeMB=34, storefileSizeMB=34, compressionRatio=1.0000, memstoreSizeMB=0, storefileIndexSizeMB=0, readRequestsCount=82887, writeRequestsCount=37063, rootIndexSizeKB=43, totalStaticIndexSizeKB=20, totalStaticBloomSizeKB=32, totalCompactingKVs=33915, currentCompactedKVs=33915, compactionProgressPct=1.0, coprocessors=[]
```

```
0 dead servers
```

```
Aggregate load: 0, regions: 8
```

*Note: the number of regions increased by four to account for the four regions in the your table.*

Scan the `hbase:meta` table again:

```
hbase> scan 'hbase:meta'
```

*Note that the `hbase:meta` information for the table has multiple regions and those regions have start and end row keys. Look at how HBase took the splits in the table creation command and made regions out of them.*



#### 5. Drop the Table

So now drop your table:

```
hbase> disable '[some name]'
```

```
hbase> drop '[some name]'
```



#### 6. Flushing Tables

Put some data into hbase:

```
hbase> create 'movie', 'cf1'
hbase> create 'location', 'cf1'
hbase> put 'movie', 1001, 'cf1:title', 'All The Kings
Men'
hbase> put 'movie', 1001, 'cf1:title', 'Ulysses'
hbase> put 'movie', 1001, 'cf1:title', 'The Sting'
hbase> put 'movie', 1001, 'cf1:title', 'When We Were Y
oung'
hbase> put 'location', 1012, 'cf1:city', 'Los Angeles'
hbase> put 'location', 1034, 'cf1:city', 'London'
hbase> put 'location', 1056, 'cf1:city', 'Benhru'
hbase> put 'location', 1052, 'cf1:city', 'Nashik'
```

Flush the movie and tags tables. This writes out the data from the memstore to HDFS:

```
hbase> flush 'movie'
hbase> flush 'location'
```

Now scan the tables:

```
hbase> scan 'movie'
hbase> scan 'location'
```



Quit the HBase shell:

```
hbase> quit
```



<h4>7. Look at the table Information</h4>

Get the full path in HDFS where the movie table's info column family's data is stored:

```
hdfs dfs -ls -R /apps/hbase/data/data/default
hdfs dfs -ls -R /apps/hbase/data/data/default/movie/*/
*info
```

Using the path from the ls, run the following command and replace the movie path with the previous command's path output:

```
hbase hfile --printkv --file /apps/hbase/data/data/default/<moviepath>
```

Results:

```
2015-03-05 18:44:00,210 INFO [main] Configuration.depr
```

ecation: hadoop.native.lib is deprecated. Instead, use io.  
native.lib.available

2015-03-05 18:44:00,580 INFO [main] util.ChecksumType:  
Checksum using org.apache.hadoop.util.PureJavaCrc32

2015-03-05 18:44:00,581 INFO [main] util.ChecksumType:  
Checksum can use org.apache.hadoop.util.PureJavaCrc32C

2015-03-05 18:44:02,680 INFO [main] Configuration.depr  
ecation: fs.default.name is deprecated. Instead, use fs.d  
efaultFS

2015-03-05 18:44:03,056 INFO [main] hfile.CacheConfig:  
Allocating LruBlockCache with maximum size 396.7 M

K: 1/info:average/1425593979703/Put/vlen=4/mvcc=0 V: 4  
.15

K: 1/info:count/1425593979703/Put/vlen=4/mvcc=0 V: 207  
7

K: 10/info:average/1425593979703/Put/vlen=4/mvcc=0 V:  
3.54

K: 10/info:count/1425593979703/Put/vlen=3/mvcc=0 V: 88  
8

K: 100/info:average/1425593979703/Put/vlen=4/mvcc=0 V:  
3.06

K: 100/info:count/1425593979703/Put/vlen=3/mvcc=0 V: 1  
28

K: 1000/info:average/1425593979703/Put/vlen=4/mvcc=0 V  
: 3.05

K: 1000/info:count/1425593979703/Put/vlen=2/mvcc=0 V:  
20

K: 1002/info:average/1425593979703/Put/vlen=4/mvcc=0 V

```
: 4.25
```

```
K: 1002/info:count/1425593979703/Put/vlen=1/mvcc=0 V:
```

```
8
```

```
K: 1003/info:average/1425593979703/Put/vlen=4/mvcc=0 V
```

```
: 2.94
```

```
...
```

The above command allows you to see how HBase stores the HFiles. All row keys are stored in sorted order; and, for each row key, all column descriptors are stored in sorted order.

*Note: The preceding screen shot shows an output labeled mvcc. HBase maintains ACID semantics using Multiversion Concurrency Control (MVCC). MVCC as implemented in HBase enables updates to occur without impacting readers, and without the need to use read locks. With MVCC, old data is not overwritten. Instead, the old data is marked as obsolete once the new data has been added.*

An mvcc value of zero means that all of the file contents can participate in any ongoing transaction.

In the case of an ongoing scan or read, a just-flushed update can be added to the ongoing read. However, if the mvcc has a higher value than that of the ongoing read, the update is not sent to the Client.

Get the full path in HDFS where the user table's info column family's data is stored:

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

Using the path from the ls, run the following command and replace the user path with the previous command's path output:

```
hbase hfile --printkv --file /hbase/data/default/user/  
<userpath>
```

The above command allows you to see how HBase stores the HFiles. All row keys are stored in sorted order and all Column Descriptors are stored in sorted order.

  
<h4>8. clean</h4>

And to get rid of everything:

```
$ bin/hbase clean
```

*Note: beware!*

```
Usage: hbase clean (--cleanZk|--cleanHdfs|--cleanAll)
```

Options:

```
--cleanZk    cleans hbase related data from zookeeper.  
--cleanHdfs  cleans hbase related data from hdfs.  
--cleanAll   cleans hbase related data from both zookeeper and hdfs.
```

## Results

Well, you did it! You can see where the command line gives you lots of usability.