Tuto1 Ibm

Version 1.0

2019-02-01

DATASOFT CONSULTING

Historique des modifications du document

|  |  |  |  |
| --- | --- | --- | --- |
| **Date** | **Version** | **Description** | **Auteur** |
| 2019-02-01 | 1.0 | Tuto1\_Ibm | Abdelaziz HANI |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |

Table des matières

1. Introduction 5

2. About this Lab 5

3. Environment Setup Requirements 5

3.1 Getting Started 5

4. Hadoop Administration 8

4.1 HDFS Disk Check 8

5. Exploring Hadoop Distributed File System (HDFS) 9

5.1 Using the command line interface 9

6. Launching the BigInsights Web Console 12

7. Working with Files in Web Console 13

8. Summary HDFS 17

9. Working with IBM InfoSphere BigInsights and Apache HBase 17

9.1 Background 17

9.2 HBase Server 18

9.3 Storing and querying data 19

9.4 Sample data and application 23

9.5 Cardinality 23

9.6 Offloading data from a warehouse table into Hbase 24

9.6.1 One-to-one mapping 24

# Introduction

The overwhelming trend towards digital services, combined with cheap storage, has generated massive amounts of data that enterprises need to effectively gather, process, and analyze. Techniques from the data warehousing and high-performance computing communities are invaluable for many enterprises. However, often times their cost or complexity of scale-up discourages the accumulation of data without an immediate need. As valuable knowledge may nevertheless be buried in this data, related scaled-up technologies have been developed. Examples include Google’s MapReduce, and the open-source implementation, Apache Hadoop.

Hadoop is an open-source project administered by the Apache Software Foundation. Hadoop’s contributors work for some of the world’s biggest technology companies. That diverse, motivated community has produced a collaborative platform for consolidating, combining and understanding data.

Technically, Hadoop consists of two key services : data storage using the Hadoop Distributed File System (HDFS) and large scale parallel data processing using a technique called MapReduce

# About this Lab

After completing this hands-on lab, you’ll be able to :

* Use Hadoop commands to explore the HDFS on the Hadoop system
* Use Infosphere BigInsights Web console to explore the HDFS on the Hadoop system

# Environment Setup Requirements

To complete this lab you will need the following :

1. InfoSphere BigInsights Bootcamp VMware® image

2. VMware Player 2.x or VMware Workstation 5.x or later

For help on how to obtain these components please follow the instructions specified in VMware Basics and Introduction from module 1.

## Getting Started

To prepare for the contents of this lab, you must go through the process of getting all of the Hadoop components started.

1. Start the VMware image by clicking the button in VMware Workstation if it is not already on.
2. Log in to the VMware virtual machine using the following information

* User : root
* Password : passw0rd

The first time you start the image a startup script might ask you a couple questions, please use the defaults for this.

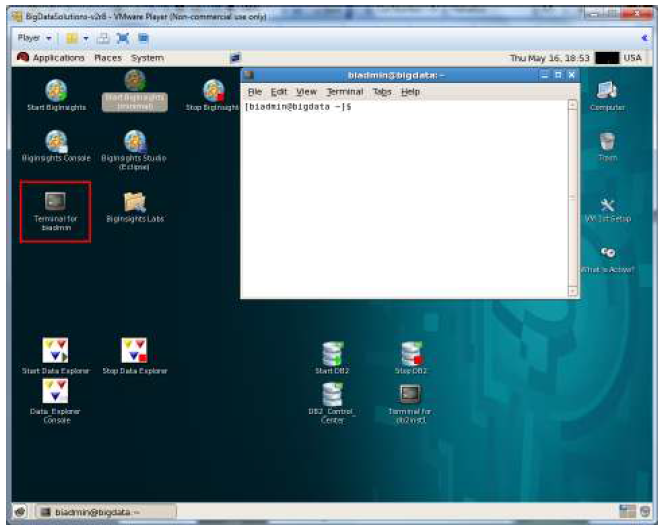
You will be asked to login a second time after the script has run. This will not always be the case if your class

manager has already set up the images.

All userids in the image we have the password “passw0rd” (the o being replaced with a zero), We will use the root user to login but we will be mostly working with the BigInsights Administrator biadmin. When you are supposed to open a Terminal you should always use the user biadmin. There is a dedicated button for this.

3. Open Command Prompt Window selecting the “Terminal for biadmin” button

# 



4. Change to the **$BIGINSIGHTS\_HOME** (which by default is set to /opt/ibm/biginsights).

**cd $BIGINSIGHTS\_HOME/bin**

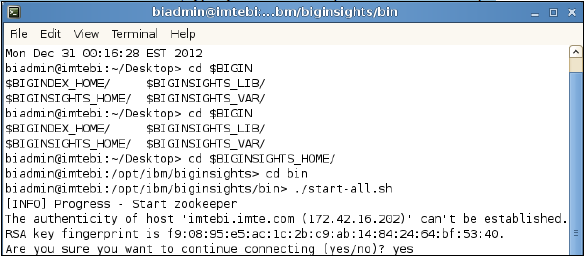
or

**cd /opt/ibm/biginsights/bin**

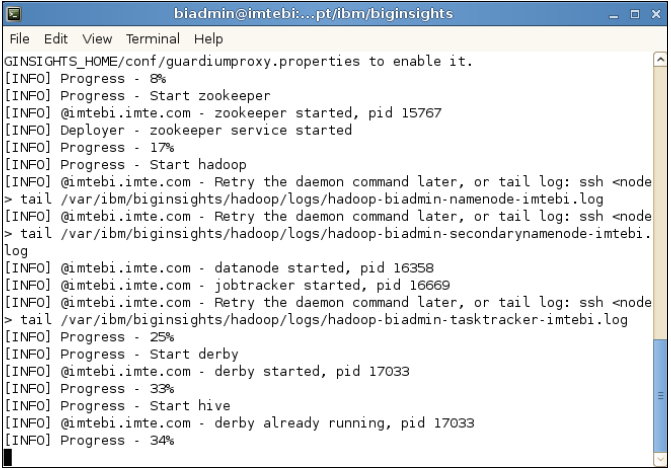
5. Start the Hadoop components (daemons) on the BigInsights server. You can practice starting all components with these commands. Please note they will take a few minutes to run :

**./start-all.sh**

If you are prompted to authenticate the host, type ‘yes’ in order to proceed with the script:



The following figure shows the different Hadoop components starting.



6. Sometimes certain hadoop components may fail to start. You can start and stop the failed components one at a time by using **start.sh** or **stop.sh** respectively. For example, to start and stop Hadoop use :

**./start.sh hadoop**

**./stop.sh hadoop**

In the following example, the console component failed. The particular component was then started again using

the **./start.sh console** command. It then succeeded without any problems. This approach can be

used for any failed components. Once all components have started successfully, you can then move to the next section.



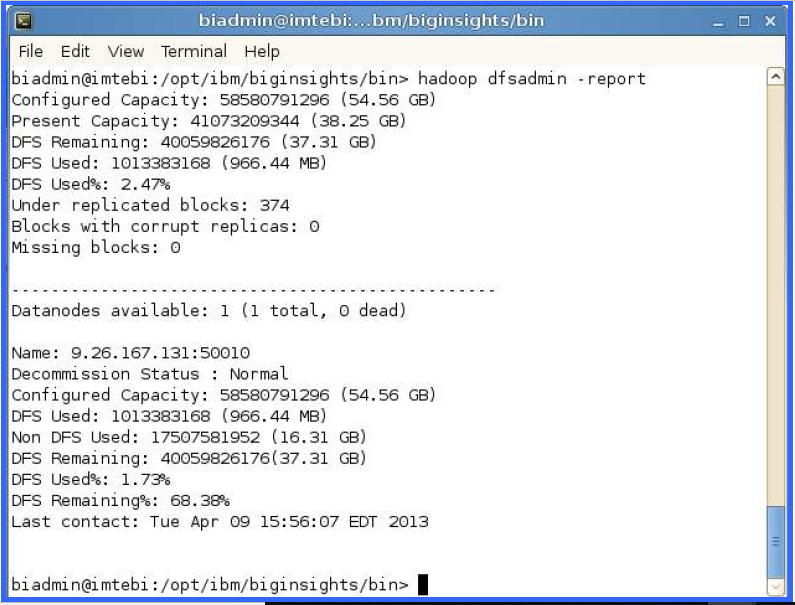
# Hadoop Administration

## HDFS Disk Check

There are various ways to monitoring the HDFS Disk, and this should be done occasionally to avoid space issues which can arise if there is low disk storage remaining. One such issue can occur if the “hadoop healthcheck” or heartbeat as it is also referred to sees that a node has gone offline. If a node is offline for a certain period of time, the data that the offline node was storing will be replicated to other nodes (since there is a 3 node replication, the data is still available on the other 2 nodes). If There is limited disk space, this can quickly cause an issue.

1. You can quickly access the HDFS report by executing the following command :

**hadoop dfsadmin -report**



# Exploring Hadoop Distributed File System (HDFS)

Hadoop Distributed File System (HDFS), allows user data to be organized in the form of files and directories. It provides a command line interface called *FS shell* that lets a user interact with the data in HDFS accessible to Hadoop MapReduce programs.

There are two methods to interact with HDFS :

1. You can use the command-line approach and invoke the FileSystem (fs) shell using the format: hadoop fs <args>.

This is the method we will use first.

2. You can also manipulate HDFS using the BigInsights Web Console. You will explore the BigInsights Web Console later

## Using the command line interface

In this part, we will explore some basic HDFS commands. All HDFS commands start with ***hadoop*** followed by ***dfs*** (distributed file system) or ***fs*** (file system) followed by a dash, and the command. Many HDFS commands are similar to UNIX commands.

For details, refer to the *Hadoop Command Guide* and *Hadoop FS Shell Guide*.

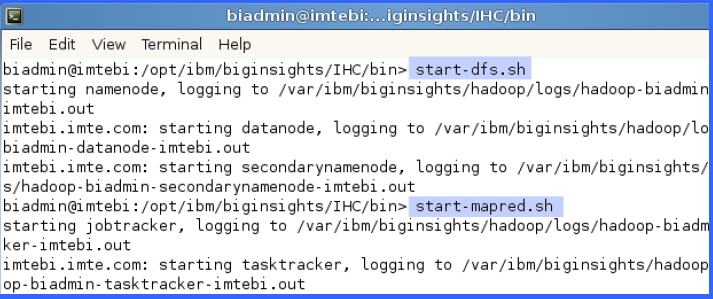
Before we can explore some HDFS commands we must ensure that the HDFS daemons are running for the cluster. The **startdfs.**

**sh** script is responsible for starting the HDFS daemons. We’ll also verify that the MapReduce daemons are running by executing the **start-mapred.sh** script ; this is required for later sections.

3. Start the HDFS daemons and the MapReduce daemons by executing the following commands :

**start-dfs.sh**

**start-mapred.sh**



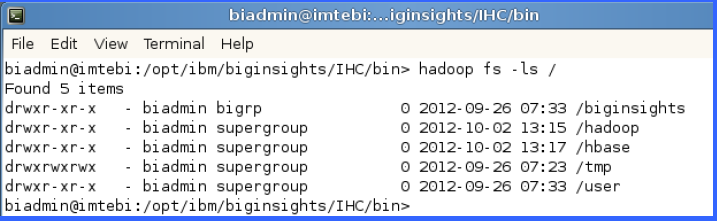
We will start with the **hadoop fs –ls** command which returns the list of files and directories with permission information.

Ensure the Hadoop components are all started, and from the same Gnome terminal window as before (and logged on as

*biadmin*), follow these instructions:

4. List the contents of the root directory.

**hadoop fs -ls /**



5. To list the contents of the /user/biadmin directory, execute:

**hadoop fs -ls**

or

**hadoop fs -ls /user/biadmin**

Note that in the first command there was no directory referenced, but it is equivalent to the second command where /user/biadmin is explicitly specified. Each user will get its own home directory under /user. For example, in the case of user biadmin, his home directory is /user/biadmin. Any command where there is no explicit directory specified will be relative to the user’s home directory.



6. To create the directory ***myTestDir*** you can issue the following command:

**hadoop fs -mkdir myTestDir**

Where was this directory created? As mentioned in the previous step, any relative paths will be using the user’s home directory.

7. Issue the ls command again to see the subdirectory myTestDir:

**hadoop fs -ls**

or

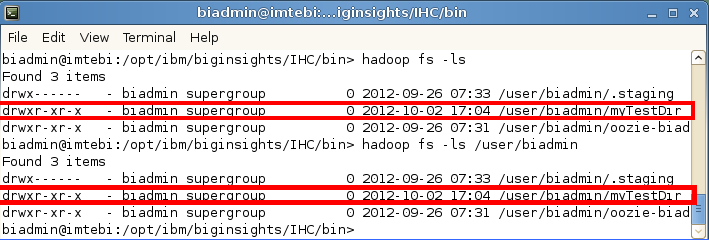
**hadoop fs –ls /user/biadmin**

To use HDFS commands recursively generally you add an “r” to the HDFS command (In the Linux shell this is generally done with the “-R” argument).

8. For example, to do a recursive listing we’ll use the –lsr command rather than just –ls, like the examples below:

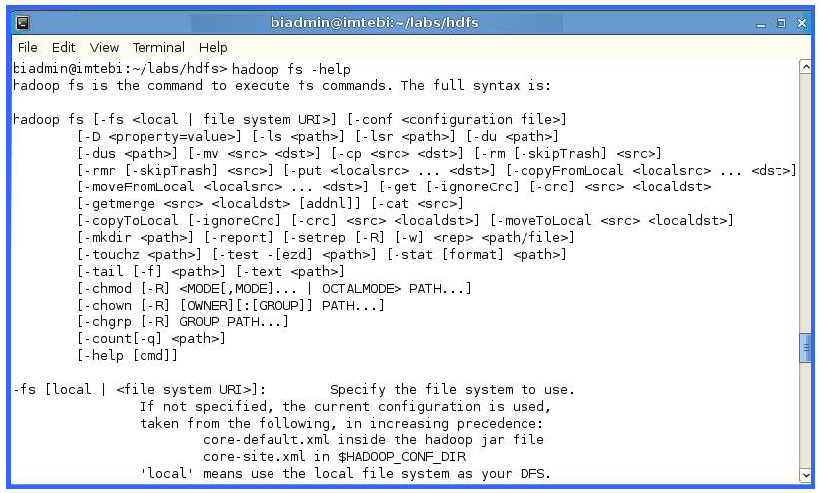
**hadoop fs -ls /user**

**hadoop fs -lsr /user**



If you would like to get more information about hadoop fs commands, invoke –help as follows:

**hadoop fs –help**



# Launching the BigInsights Web Console

1. Launching the web console is done by entering a URL into a web browser. The format for the URL is:

http://<host>:<port> **or** http://<host>:<port>/data/html/index.html

The default is:



For convenience, there is a shortcut on the biadmin's user Desktop, which will launch the web console when doubleclicked. It is called BigInsights Console.

2. If you use the link you will already have the username and password entered. If you enter the URL the login credentials are

* User: biadmin
* Password: passw0rd

3. Verify that the BigInsights Web Console looks like this :



# Working with Files in Web Console

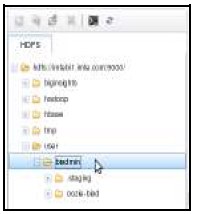
The **Files** tab of the console enables you to explore the contents of your file system, create new subdirectories, upload small files for test purposes, and perform other file-related functions. In this module, you’ll learn how to perform such tasks against the Hadoop Distributed File System (HDFS) of BigInsights.

Examples in this section use the **biadmin** user which has a **/user/biadmin** directory in its distributed file system. If you're accessing a BigInsights cluster using a different user ID, adapt the instructions in this exercise to work with your home directory in HDFS.

1. Click the **Files** tab of the console to begin exploring the distributed file system.



1. Expand the directory tree shown in the pane at left. If files have been uploaded to HDFS before, you’ll be able to navigate through the directory to locate them. In our case, the figure below shows **/user/biadmin** containing default subdirectories.

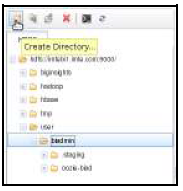


1. Become familiar with the functions provided through the icons at the top of this pane, as we'll refer to some of these in subsequent sections of this module. Simply hover your cursor over the icon to learn its function.From left to right, the icons enable you to create a directory, upload a file to HDFS, download a file from HDFS to your local file system, delete a file from HDFS, open a command window to launch HDFS shell commands, and refresh the Web console page.

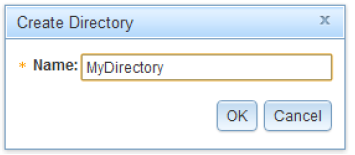


1. Position your cursor on the **user/biadmin** directory and click the **Create Directory** icon (at far left) to

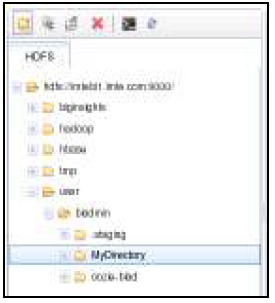
create a subdirectory for test purposes



1. When a pop-up window appears prompting you for a directory name, enter **MyDirectory** and click **OK.**



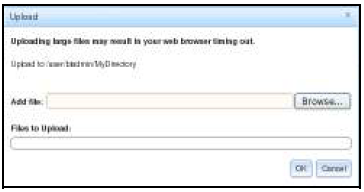
1. Expand the directory hierarchy to verify that your new subdirectory was created.



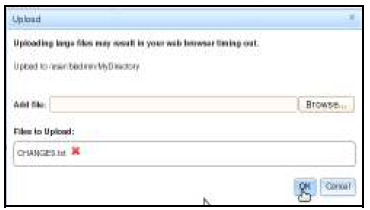
1. Click the **Upload** button to upload a small sample file from your local file system to HDFS.



1. When the pop-up window appears, click the **Browse** button to browse your local file system for a sample file.

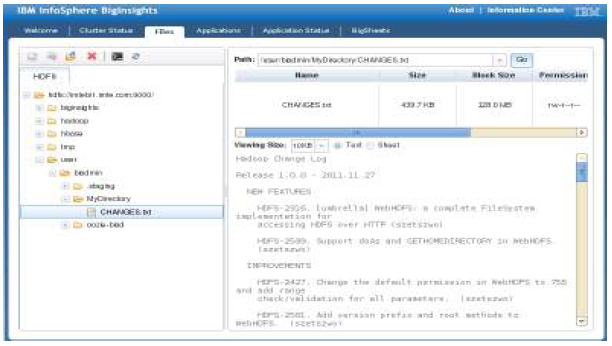


1. Navigate through your local file system to the directory ***biadmin/labs/hdfs*** and select the ***CHANGES.txt*** file. Click **OK.**
2. Verify that the window displays the name of this file. Note that you can continue to click **Choose file** for additional files to upload and that you can delete files as upload targets from the displayed list. However, for this exercise, simply click



1. When the upload completes, verify that the CHANGES.txt file appears in the directory tree at left. On

the right, you should see a subset of the file’s contents displayed in text format.



1. Let’s now download the same file from HDFS to your local file system. Highlight the ***CHANGES.txt***

file in your ***MyDirectory*** directory and click the **Download** button. (This button is between the **Upload** and **Delete** buttons.)



1. When prompted, click the **Save File** button. Then select OK.



When prompted, identify the folder (directory) of your local file system where the file should be stored. Wait until the Web browser indicates that the file download is complete.

# Summary HDFS

You have just completed Lab 1 which focused on the basics of the Hadoop platform, including HDFS and a quick introduction to

the web console. You should now know how to perform the following basic tasks on the platform:

* Start/Stop the Hadoop components
* Interact with the data in the Hadoop Distributed File System (HDFS)
* Change Hadoop configuration values.
* Navigate within HDFS using both Hadoop commands and BigInsights Web Console.

# Working with IBM InfoSphere BigInsights and Apache HBase

## Background

BigInsights Enterprise Edition provides several options to access HBase such as :

1. HBase shell, which is available stand-alone and also available inside Eclipse environment.

2. Jaql HBase module, bundled with Jaql, which is a query language with a data model based on JavaScript ObjectNotation (JSON).

3. HBase application, accessible through the BigInsights Web Console. (Before you can launch this application, an administrator must deploy it on your BigInsights cluster and authorize you to access the application.)

4. BigSQL client and JDBC driver which provide SQL access to data residing on HDFS including HBase.

In this lab, you’ll use options 1-3 to store and query data that has been offloaded from a sample warehouse into HBase.GSDB database, a rich and realistic database that contains sample data for the Great Outdoors company, which is a fictional outdoor equipment retailer is used for this purpose. For simplicity, we will use only a few tables from this database.

While HBase can be used for a wide variety of data storage scenarios, it is also being used as an alternative for datawarehouses. The lab covers various design options and provides information to compare them.

Before diving into design considerations, you need to be familiar with running and monitoring HBase server and basic operations.

## HBase Server

section, we will briefly go over the user interface which is part of BigInsights Web Console.

1. Log into BigInsights web console and open the *Cluster Status* tab. Select HBase to view the status of Hbase master and region servers.



Make sure to start the HBase server if it is stopped.

2. Use Master and RegionServer web interfaces to visualize tables, regions and other metrics. By default, Master web interface is started on port 60010; RegionServer on 60030.

Master web interface: http://bigdata:60010

RegionServer web interface: http://bigdata:60030

Some interesting information from the web interfaces which we will check later in the lab are :

* HBase root directory – This can be used to find the size of an HBase table.
* List of tables with descriptions.
* For each table, it displays the list of regions with start and end keys. This information can be used to compact or split tables as needed.
* Metrics for each region server. These can be used to determine if there hot regions which are serving the majority of requests to a table. Such regions can be split. It also helps determine the effects and effectiveness of block cache, bloom filters and memory settings. Based on usage, region servers can be added/removed using the commands :

$BIGINSIGHTS\_HOME/bin/addnode.sh hbase <node1> <node2>… -type=<nodetype>

$BIGINSIGHTS\_HOME/bin/removenode.sh hbase <node1> <node2>… -type=<nodetype>

where node1, node2 etc can be IP address or host name and type can be regionserver or master. BigInsights allows specifying more than one master for handling failover. However, for this lab, only one master will be active.

3. Perform a health check of HBase which is different from the status checks done above. It verifies the health of the functionality.

4. Launch a console as biadmin (you can find a link on the Linux desktop)

$BIGINSIGHTS\_HOME/bin/healthcheck.sh hbasez

## Storing and querying data

Before creating tables that correspond to the warehouse data, we will first look at creation of simple table and understand how HBase stores data.

1. Launch hbase shell using the command :



1. Create an HBase table with three column families by using the *create* command :

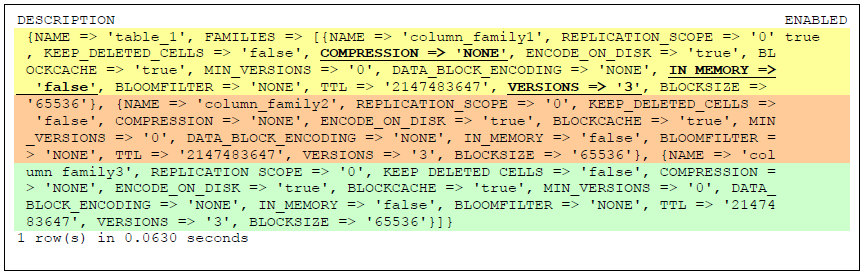


Tables in HBase are made of rows and columns. All columns in HBase belong to, and are grouped into a particular column family. From the command above you will notice that we have specified 3 column families associated with table\_1.

The create command only requires the name of table and one or more column families. Column families must be declared up front at schema definition time whereas columns can be added dynamically to the table. Each row can also have a different set of columns.

It is important to note that all column family members are stored together on the filesystem. The reason this is relevant will become apparent in the following steps.

Check the default properties used for column families using *describe command :*



As you can see here, there are several properties that are configurable for each column family created. We won’t go into every single property, but over the next couple set of steps, we will look at modifying the highlighted options for *column\_family1.*

1. Specify compression for a column family in the table. For HBase packaged with BigInsights, only gzip compression can be used out of the box. Use *alter* command. To alter a table, it has to be disabled first.



SNAPPY and LZO are the two other compression algorithms that are supported by HBase but would need extra configuration. Note that GZIP is slower than LZO because it generally compresses better. This would add to query latency, but in some cases it better compression may be preferred.

Generally, production systems should use compression with their ColumnFamily definitions. Although, the data size will be smaller on disk, when accessed in memory for example, it will be inflated. Therefore just using compression will not eliminate the problem of over-sized column family names or over-sized column names. Care must be taken when determining the schema. We will mention some of these best practices later on as well.

1. Specify IN\_MEMORY option for a column family that will be queried frequently.



This does not ensure that all the data will always be in memory. Data will always be persisted to disk. It only gives priority for the corresponding data within the column family to stay in the cache longer.

1. Specify the required number of versions as 1 for the column family.

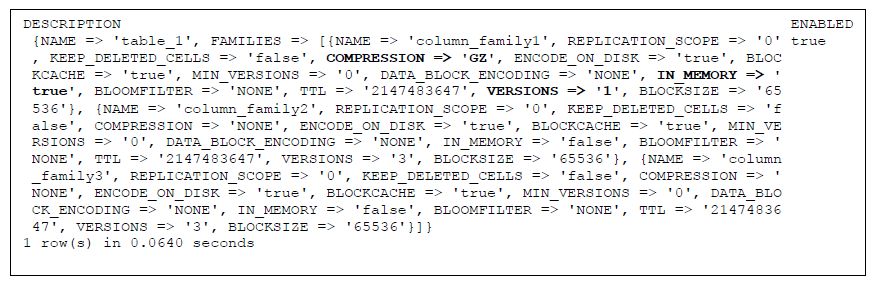


HBase table cells (tuples) are versioned. That means multiple versions of a cell can be stored. If your application does not require multiple versions, specify VERSIONS => 1. When mapping data from relational DBMS, multiple versions may not be required. By default, 3 versions of a value will be stored.

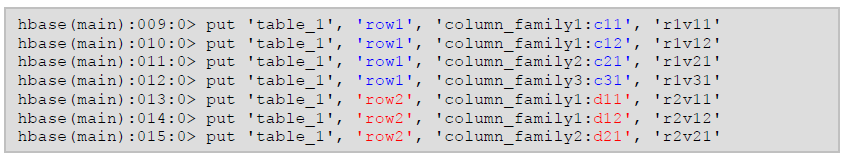
It is possible to have multiple cells where the row keys and columns are identical and only differs in its version dimension. Versions are specified using a long integer, i.e., timestamp.

1. Run *describe against the table* again to verify the above changes. Before you can load data, you

will have to *enable* the table.

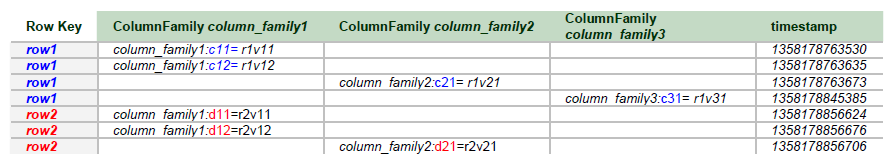


1. Insert data using put command. As mentioned earlier, each row can have a different set of columns. Below are a set of put commands which will insert two rows with a different set of column names.



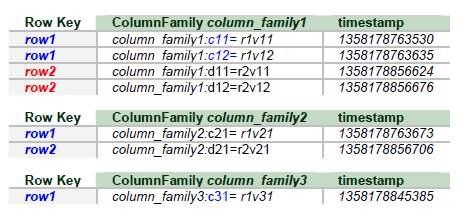
The following figure shows a strictly CONCEPTUAL view of table\_1:

From a purely conceptual perspective, tables may be viewed as a sparse set of rows as shown above. However, physically they are stored on a per-column family basis as mentioned previously.



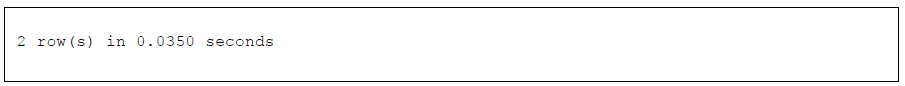
It is very important to note in the diagram above that the empty cells are not stored since they need not be in a column-oriented storage format. This is different from RDBM’s in the sense HBase doesn’t store nulls.

The physical view is as follows :

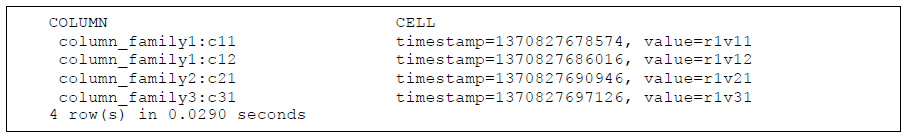


1. Verify two rows have been inserted using the *count* command. The *count* command works for

small tables. For much larger tables, you can use the RowCounter mapreduce job which we will use later in the lab.



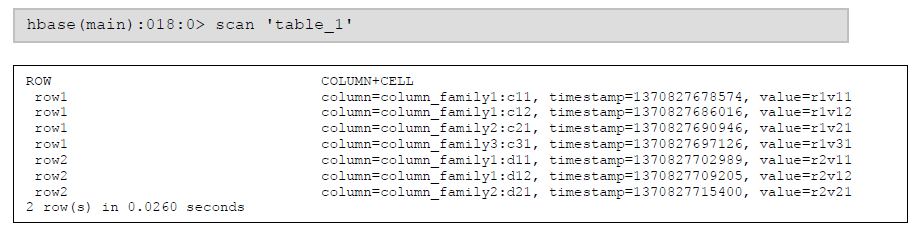
1. To view the data, you may use *get,* which returns attributes for a specified row, or *scan* which allows iteration over multiple rows.



In the above example, the 4 rows correspond to the values of 4 columns for row1.

1. Now, run the scan command to see a per-row listing of values.

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.



Note that, in addition to the actual column value (*r1v11*), each result row has the row key value (*row1*), column family name (*column\_family1*), 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.

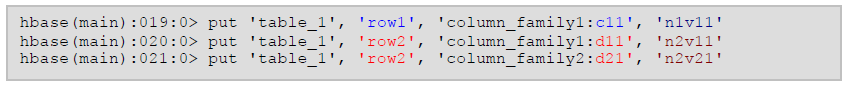
Tips :

Try to use smaller row key values, column family and qualifier names.

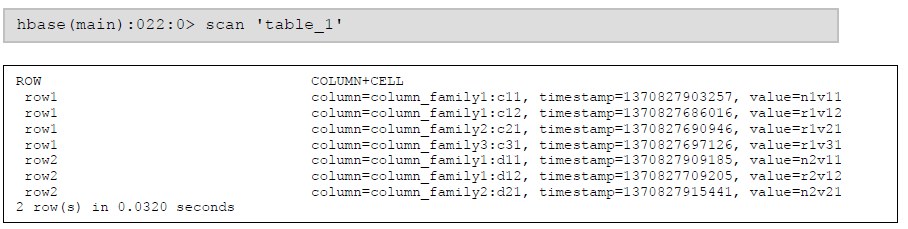
Try to use fewer columns if you have dense data.

1. HBase does not have an *update* command or API. An *update* is same as *put* with another set of

column values for the same row key. Update values in column\_family1 (with VERSIONS => 1) and column\_family2 (with VERSIONS => 3).



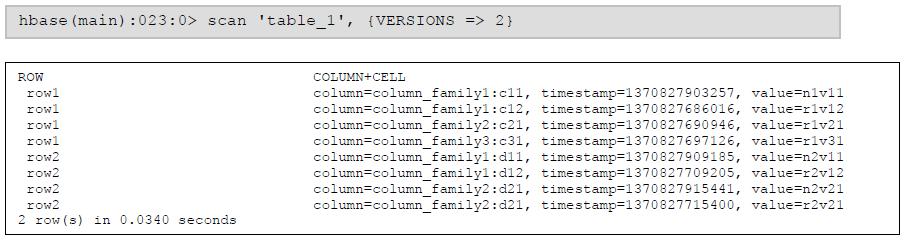
A scan will show rows updated values.



Notice the old versions of the cells are not shown in this result set. By default, if you don’t specify an explicit version when doing a scan, the cell(s) whose version has the largest value will be returned.

1. View values with different versions stored for column\_family2 by explicitly requesting multiple

versions.

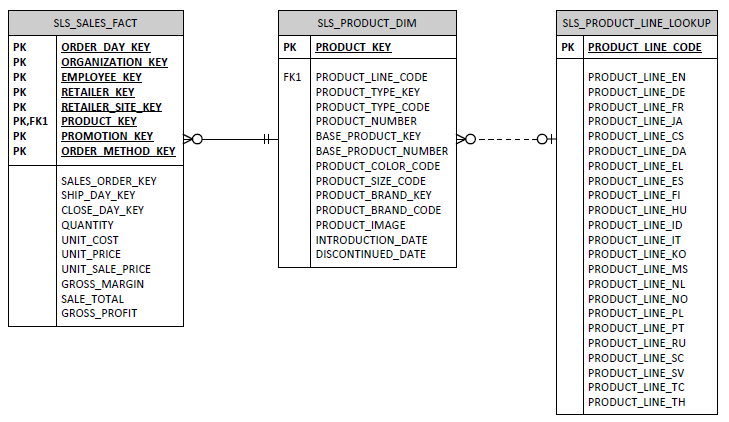


The first row in the result for row2|column\_family2: d21 is the new value and the second row is the old value. Don’t forget that column family 1 versioning was set to 1. Therefore, we only see the latest (and only) version for that column family.

## Sample data and application

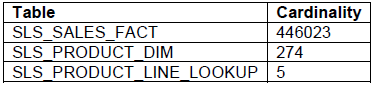
In this lab, we will use three tables from GSDB’s GOSALESDW database.

* SLS\_SALES\_FACT
* SLS\_PRODUCT\_DIM
* SLS\_PRODUCT\_LINE\_LOOKUP



## Cardinality

Another aspect to consider in the design is the size of the each table.



For checking how the database performs with bigger dataset, we will check with a scale of 10

## Offloading data from a warehouse table into Hbase

Among the numerous options that HBase provides to load data, bulk loading is very useful for off loading warehouse data. In such scenarios, an administrator would have exported the warehouse data into some delimited format. Let us assume,

We already have data files in tsv format from the relations mentioned in the previous section. To

load this data into HBase, we can use the ImportTsv and CompleteBulkLoad utilities provided by HBase.

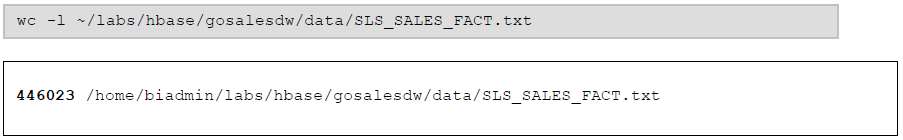
ImportTsv is a utility that will load data in TSV format into HBase. It has two distinct usages: (A) loading data from TSV format in HDFS into HBase via Puts, and (B) preparing StoreFiles to be loaded via the completebulkload.Let us try this for one table SLS\_SALES\_FACT to see how it works, first loading data via Puts (non-bulk load), and than later using StoreFiles with CompleteBulkLoad (bulk load).

In this section, we will also try to understand how HBase handles row keys and some pitfalls that users may encounter when moving data from relational database to HBase tables. We will also try some useful options like pre-creating regions to see how it can help with data loading and queries.

### One-to-one mapping

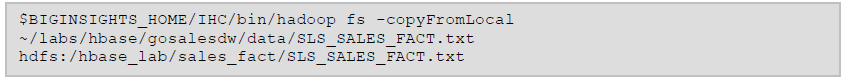
In this step, we will use a one-to-one mapping of the columns in a relational table to an HBase table row key and columns. This is not a recommended approach. The goal of this exercise is to demonstrate the inefficiency and pitfalls that can occur with such a mapping.

1. Examine the number of lines (rows) in the data file. (You need to quit the HBase console with ‘quit’ or open up another linux terminal).



We would expect to have same number of rows once in HBase.

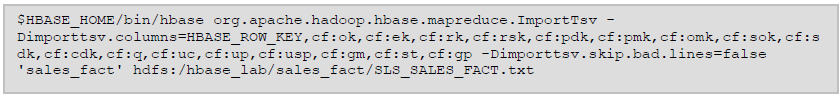
2. Copy data file to be loaded into HBase onto hdfs:



3. Create a table sales\_fact with a single column family that stores only one version of its values.

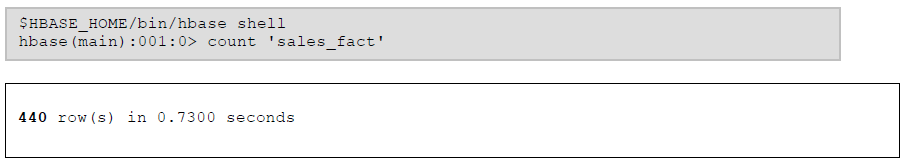


4. Use ImportTsv tool to load data. Note this step may take several minutes.



As mentioned previously this ImportTsv operation loads data into HBase via Puts (i.e., non-bulk loading):

Now,count the rows in the results



Observations

* There were no errors during the load but there it is obviously apparent that there are fewer rows in the Hbase table than were in the original file. This is because HBase treats updates (i.e., versioning) differently. Hbase enforces a unique row key. However, if a new put request (insert) comes in with same row key value, the row is updated. This is different from most relational database systems where users expect an error in such scenarios. This makes it important to ensure the rows have highly unique key values.