

Tutorial:

Introduction to HBase

Imani Palmer

Last update: September 14, 2015

1 Overview

Welcome

Welcome to the HBase tutorial. This tutorial covers the critical skills needed to develop a Hadoop HBase application. It starts with an already deployed Hadoop environment where students will execute a series of hands-on labs.

Objectives

Upon completing this tutorial, students will be able to:

- Set up an all-in-one HBase installation
- Perform basic operations on HBase
- Develop simple HBase applications
- Compile HBase applications
- Run HBase applications and examine the output

Structure

This guide is designed as step-by-step instructions for hands-on exercises. It teaches you, through the use of examples, how to operate the HBase service in a functional test environment. The instructions are presented in the following order:

1. Download and set up an all-in-one HBase VM
2. Use Java to interface with HBase
3. Run applications on HBase and examine the output

2 Requirements

This tutorial is designed to work on the **Hortonworks Sandbox 2.3** virtual machine. In order to complete the tutorial, you need to have a working HortonWorks Sandbox machine either locally or on the Amazon Web Services.

All assignments are designed based on **JDK 7** (included in the virtual machine).



Please refer to **Tutorial: Run HortonWorks Sandbox 2.3 Locally** or **Tutorial: Run HortonWorks Sandbox 2.3 on AWS** for more information.

3 Setup Hadoop Virtual Machine

Step 1: Start the virtual machine, and then connect to it through the SSH.

Step 2: After successfully logging in, you should see a prompt similar to the following:

```
[root@sandbox ~]#
```

Step 3: Start HBase Service:

```
# bash ~/start_hbase.sh
```

Step 4: Create a new folder for this tutorial:

```
# mkdir hbase-tutorial
```

Step 5: Change the current folder:

```
# cd hbase-tutorial
```

4 Create Table

The “Create Table” application is your first introduction in to interface with HBase. It is a straightforward application in which you will create the table named “employee” shown below:

Row key	personal data	professional



If you are new to Java programming language, take a look at:
<https://docs.oracle.com/javase/tutorial/>

Step 1: Create a new Java source code file using the following command:

```
# nano CreateTable.java
```

Step 2: Type (or copy/paste) the Java code “**Appendix A**” in the editor, then quit **nano**, and save the file.



To learn more about developing HBase applications in Java, visit:
<http://hbase.apache.org/book.html>

Step 2 (Alternative): Quit **nano**. Download the source from the github:

```
# wget  
https://github.com/xldrx/hbase_examples/raw/master/tutorial/CreateTable.java
```

Step 3: Compile the source code using the following commands.

```
# export CLASSPATH=$(hbase classpath)  
# javac CreateTable.java
```

Step 4 (Optional: Just in case of compile errors): If you encounter any compilation error (possibly due to typos), you need to first edit the source file and fix the problems, then clean the built files, and finally compile the code again using the direction above. In summary, these are the commands:

```
# nano CreateTable.java  
# rm -rf *.class  
# export CLASSPATH=$(hbase classpath)  
# javac CreateTable.java
```

Step 5: Run the application using the following command and wait for it to be done.

```
# java CreateTable
```

The output should contain the following:

```
Table created
```

Step 6 (Optional: Just in case of runtime error): If you get “client.RpcRetryingCaller: Call exception”, restart the HBase service using the following command and run the application again.

```
# service hbase-starter restart
```

Step 8 (Optional: Just in case of other runtime errors): If you encounter any compilation error (possibly due to typos), you need to first edit the source file to fix the problems, then compile the code again using the direction above.

Additionally, the created table should be removed before a new run by using following command:

```
# hbase shell
> disable 'emp'
> drop 'emp'
> exit
```

5 List Table

Now that you have created a table, you also want to see what you have created. You will now write an application that will show you the tables listed in HBase.

Step 1: Create a new Java source code file using the following command:

```
# nano ListTables.java
```

Step 2: Type (or copy/paste) the Java code of “**Appendix B**” in the editor, then quit **nano**, and save the file.

Step 2 (Alternative): Quit **nano**. Download the source from the github:

```
# wget
https://github.com/xldrx/hbase_examples/raw/master/tutorial/ListTables.java
```

Step 3: Compile and execute the program as shown below.

```
# export CLASSPATH=$(hbase classpath)
# javac ListTables.java
# java ListTables
```

The output should show the table you created with CreateTable.java, as shown below:

```
usr
emp
```

6 Inserting Data

You have a table and you can see it, so let's put some data into your table. At the end of this section, you will have a table like the one shown below.

Row key	personal data		professional	
Empid	Name	City	Designation	Salary
1	Raju	Hyderbad	Manager	50,000
2	Ravi	Chennai	Sr. engineer	30,000
3	Rajesh	Delhi	Jr. engineer	25,000

Step 1: Create a new Java source code file using the following command:

```
# nano InsertData.java
```

Step 2: Type (or copy/paste) the Java code of “**Appendix C**” in the editor, then quit **nano**, and save the file.

Step 2 (Alternative): Quit **nano**. Download the source from the github:

```
# wget
https://github.com/xldrx/hbase_examples/raw/master/tutorial/InsertData.java
```

Step 3: Compile and execute the program as shown below.

```
# export CLASSPATH=$(hbase classpath)
# javac InsertData.java
# java InsertData
```

The output should be:

```
data inserted
```

7 Retrieve Data

You have this filled table as shown below; now you will read data from the table.

Row key	personal data		professional	
Empid	Name	City	Designation	Salary
1	Raju	Hyderbad	Manager	50,000
2	Ravi	Chennai	Sr. engineer	30,000
3	Rajesh	Delhi	Jr. engineer	25,000

Step 1: Create a new Java source code file using the following command:

```
# nano RetrieveData.java
```

Step 2: Type (or copy/paste) the Java code of “**Appendix D**” in the editor, then quit **nano**, and save the file.

Step 2 (Alternative): Quit **nano**. Download the source from the github:

```
# wget
https://github.com/xldrx/hbase_examples/raw/master/tutorial/RetrieveData.java
```

Step 3: Compile and execute the program as shown below.

```
# export CLASSPATH=$(hbase classpath)
# javac RetrieveData.java
# java RetrieveData
```

The output should be:

```
name: raju city: hyderabad
```

8 Scan Data

You will now scan to view the data in this table. Scanning allows you to get all the data from the table.

Row key	personal data		professional	
Empid	Name	City	Designation	Salary
1	Raju	Hyderbad	Manager	50,000
2	Ravi	Chennai	Sr. engineer	30,000
3	Rajesh	Delhi	Jr. engineer	25,000

Step 1: Create a new Java source code file using the following command:

```
# nano ScanTable.java
```

Step 2: Type (or copy/paste) the Java code of “**Appendix E**” in the editor, then quit **nano**, and save the file.

Step 2 (Alternative): Quit **nano**. Download the source from the github:

```
# wget
https://github.com/xldrx/hbase_examples/raw/master/tutorial/ScanTable.java
```

Step 3: Compile and execute the program as shown below.

```
# export CLASSPATH=$(hbase classpath)
# javac ScanTable.java
# java ScanTable
```

The output should be similar to:

```
Found row :
keyvalues={row1/personal:city/1442101994699/Put/vlen=9/seqid=0,
row1/personal:name/1442101994699/Put/vlen=4/seqid=0}
```

Appendix A: Create Table Source Code

```
import java.io.IOException;

import org.apache.hadoop.hbase.HBaseConfiguration;
import org.apache.hadoop.hbase.HColumnDescriptor;
import org.apache.hadoop.hbase.HTableDescriptor;
import org.apache.hadoop.hbase.client.HBaseAdmin;
import org.apache.hadoop.hbase.TableName;

import org.apache.hadoop.conf.Configuration;

public class CreateTable {

    public static void main(String[] args) throws IOException {

        // Instantiating configuration class
        Configuration con = HBaseConfiguration.create();

        // Instantiating HbaseAdmin class
        HBaseAdmin admin = new HBaseAdmin(con);

        // Instantiating table descriptor class
        HTableDescriptor tableDescriptor = new
        HTableDescriptor(TableName.valueOf("emp"));

        // Adding column families to table descriptor
        tableDescriptor.addFamily(new HColumnDescriptor("personal"));
        tableDescriptor.addFamily(new HColumnDescriptor("professional"));

        // Execute the table through admin
        admin.createTable(tableDescriptor);
        System.out.println(" Table created ");
    }
}
```


Appendix B: List Table Source Code

```
import java.io.IOException;

import org.apache.hadoop.conf.Configuration;

import org.apache.hadoop.hbase.HBaseConfiguration;
import org.apache.hadoop.hbase.HTableDescriptor;
import org.apache.hadoop.hbase.MasterNotRunningException;
import org.apache.hadoop.hbase.client.HBaseAdmin;

public class ListTables {

    public static void main(String args[])throws
MasterNotRunningException, IOException {

        // Instantiating a configuration class
        Configuration conf = HBaseConfiguration.create();

        // Instantiating HBaseAdmin class
        HBaseAdmin admin = new HBaseAdmin(conf);

        // Getting all the list of tables using HBaseAdmin object
        HTableDescriptor[] tableDescriptor = admin.listTables();

        // printing all the table names.
        for(int i=0; i < tableDescriptor.length;i++){
            System.out.println(tableDescriptor[i].getNameAsString());
        }
    }
}
```

Appendix C: Insert Data Source Code

```
import java.io.IOException;

import org.apache.hadoop.conf.Configuration;

import org.apache.hadoop.hbase.HBaseConfiguration;
import org.apache.hadoop.hbase.client.HTable;
import org.apache.hadoop.hbase.client.Put;
import org.apache.hadoop.hbase.util.Bytes;

public class InsertData{

    public static void main(String[] args) throws IOException {

        // Instantiating Configuration class
        Configuration config = HBaseConfiguration.create();

        // Instantiating HTable class
        HTable hTable = new HTable(config, "emp");

        // Instantiating Put class
        // accepts a row name.
        Put p = new Put(Bytes.toBytes("row1"));

        // adding values using add() method
        // accepts column family name, qualifier/row name ,value
        p.add(Bytes.toBytes("personal"),
            Bytes.toBytes("name"),Bytes.toBytes("raju"));

        p.add(Bytes.toBytes("personal"),
            Bytes.toBytes("city"),Bytes.toBytes("hyderabad"));

        p.add(Bytes.toBytes("professional"),Bytes.toBytes("designation"),
            Bytes.toBytes("manager"));

        p.add(Bytes.toBytes("professional"),Bytes.toBytes("salary"),
            Bytes.toBytes("50000"));

        // Saving the put Instance to the HTable.
        hTable.put(p);
        System.out.println("data inserted");

        // closing HTable
        hTable.close();
    }
}
```

Appendix D: Retrieve Data Source Code

```
import java.io.IOException;

import java.io.IOException;

import org.apache.hadoop.conf.Configuration;

import org.apache.hadoop.hbase.HBaseConfiguration;
import org.apache.hadoop.hbase.client.Get;
import org.apache.hadoop.hbase.client.HTable;
import org.apache.hadoop.hbase.client.Result;
import org.apache.hadoop.hbase.util.Bytes;

public class RetrieveData{

    public static void main(String[] args) throws IOException, Exception{

        // Instantiating Configuration class
        Configuration config = HBaseConfiguration.create();

        // Instantiating HTable class
        HTable table = new HTable(config, "emp");

        // Instantiating Get class
        Get g = new Get(Bytes.toBytes("row1"));

        // Reading the data
        Result result = table.get(g);

        // Reading values from Result class object
        byte [] value = result.getValue(Bytes.toBytes("personal"),Bytes.toBytes("name"));

        byte [] value1 = result.getValue(Bytes.toBytes("personal"),Bytes.toBytes("city"));

        // Printing the values
        String name = Bytes.toString(value);
        String city = Bytes.toString(value1);

        System.out.println("name: " + name + " city: " + city);
    }
}
```

Appendix E: Scan Data Source Code

```
import java.io.IOException;

import org.apache.hadoop.conf.Configuration;

import org.apache.hadoop.hbase.HBaseConfiguration;
import org.apache.hadoop.hbase.util.Bytes;

import org.apache.hadoop.hbase.client.HTable;
import org.apache.hadoop.hbase.client.Result;
import org.apache.hadoop.hbase.client.ResultScanner;
import org.apache.hadoop.hbase.client.Scan;

public class ScanTable{

    public static void main(String args[]) throws IOException{

        // Instantiating Configuration class
        Configuration config = HBaseConfiguration.create();

        // Instantiating HTable class
        HTable table = new HTable(config, "emp");

        // Instantiating the Scan class
        Scan scan = new Scan();

        // Scanning the required columns
        scan.addColumn(Bytes.toBytes("personal"), Bytes.toBytes("name"));
        scan.addColumn(Bytes.toBytes("personal"), Bytes.toBytes("city"));

        // Getting the scan result
        ResultScanner scanner = table.getScanner(scan);

        // Reading values from scan result
        for (Result result = scanner.next(); result != null; result = scanner.next())

            System.out.println("Found row : " + result);
        //closing the scanner
        scanner.close();
    }
}
```