Hands-On MapRDB (Binary Tables)

*Sourabh S Jain (Technical Architect) sourabh.jain@mastek.com*

*Hardik N Ashar (Software Specialist)*

*Hardik.ashar@mastek.com*

Date : 27/04/2017

Version : 0.0.1

Contents

[Introduction to MapRDB 3](#_Toc481080406)

[MapRDB and HBase 4](#_Toc481080407)

[Hands-On Code Walk Through – Java 5](#_Toc481080408)

[Preface: 5](#_Toc481080409)

[Pre-requisites: 5](#_Toc481080410)

[Steps 5](#_Toc481080411)

[Hands-On Code Walk Through – C 11](#_Toc481080412)

[References 12](#_Toc481080413)

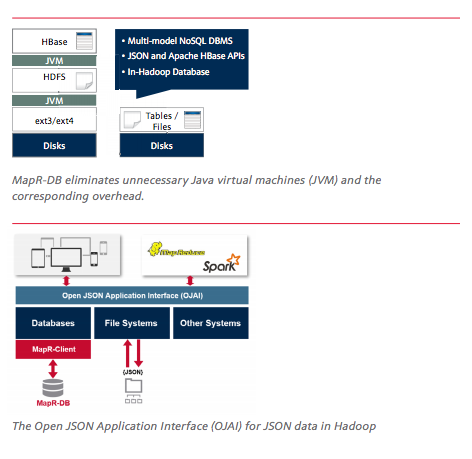
# Introduction to MapRDB

MapR-DB is an enterprise-grade, high performance, in-Hadoop NoSQL database management system. As part of the MapR Converged Data Platform, MapR-DB provides the first in-Hadoop document database that allows developers to deliver scalable applications that also leverage continuous analytics on real-time data. MapR-DB also supports a wide column data model and API to run existing Apache HBase™ applications faster and more reliably.

With MapR-DB, the MapR Platform is the only data platform built for running NoSQL operations, Hadoop analytics, and event stream processing in the same cluster.

Only MapR-DB handles high volume and high velocity database workloads alongside batch analytical tasks. This is due to the MapR architecture which uniquely enables fast, random read/write access to files and tables. MapR-DB is a multi-model NoSQL database that uses Open JSON Application Interface (OJAI™) for document database capabilities, as well as a wide column API to run existing HBase applications faster and more reliably. It manages many operational data formats including log data, sensor data, metadata, clickstreams, user profiles, session states, and link/semantics/relationship data.

The MapR Converged Data Platform also provides support for HBase as an add-on to give developers a choice of in-Hadoop databases. Developers can run applications simultaneously on HBase and MapR-DB in the same deployment.



# MapRDB and HBase

[Apache HBase](https://www.mapr.com/products/product-overview/apache-hbase) is a database that runs on a [Hadoop](https://www.mapr.com/products/apache-hadoop) cluster. HBase is not a traditional RDBMS, as it relaxes the ACID (Atomicity, Consistency, Isolation, and Durability) properties of traditional RDBMS systems in order to achieve much greater scalability. Data stored in HBase also does not need to fit into a rigid schema like with an RDBMS, making it ideal for storing unstructured or semi-structured data.

The MapR Distribution including Apache Hadoop supports HBase, but also supports [MapR-DB](https://www.mapr.com/products/mapr-db-in-hadoop-nosql), a high performance, enterprise-grade NoSQL DBMS that includes the HBase API to run HBase applications. Many of the advantages of using HBase in data architecture apply to MapR-DB. MapR built MapR-DB to take HBase applications to the next level, so if the thought of higher powered, more reliable HBase deployments sound appealing to any enterprise, take a look at some of the MapRDB use cases (<https://mapr.com/products/mapr-db-in-hadoop-nosql/> ).

# Hands-On Code Walk Through – Java

## Preface:

In this section, we will walk through the Java Code to connect to MapRDB to perform the below set of operations:

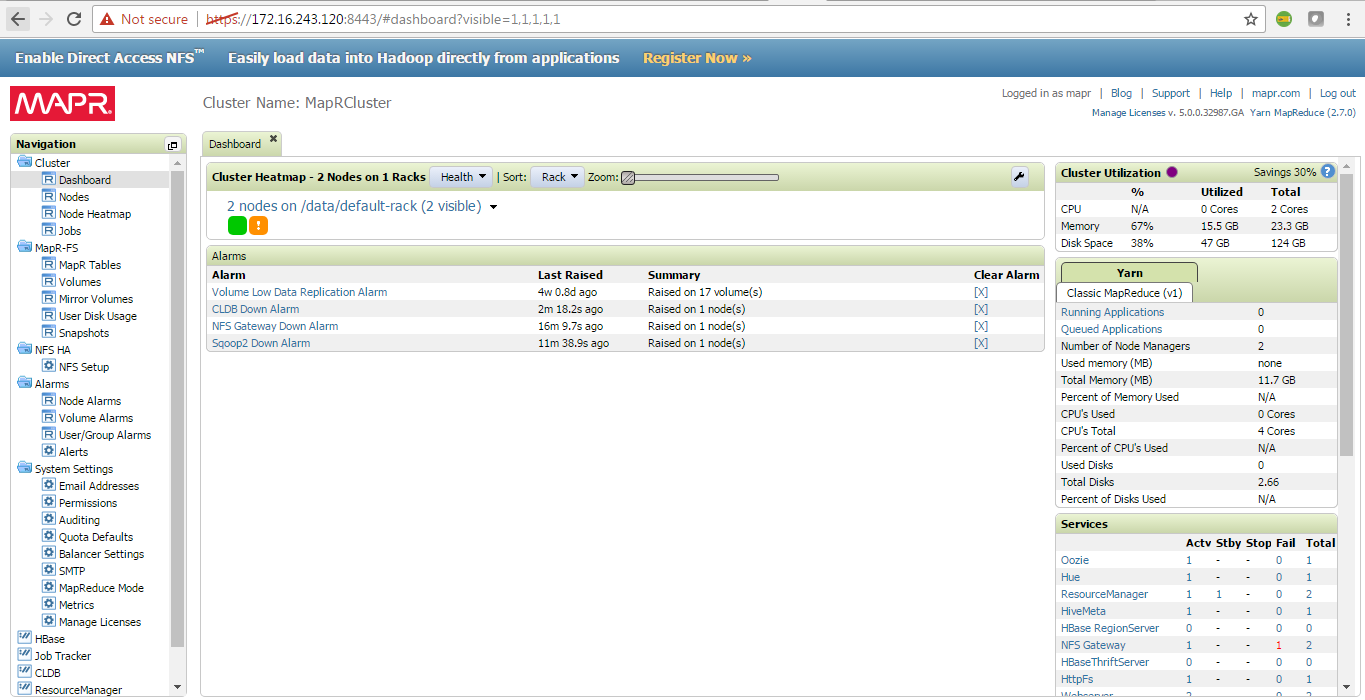
* CREATE/DROP MapR-DB Tables
* CRUD Operations on MapR-DB Tables

## Pre-requisites:

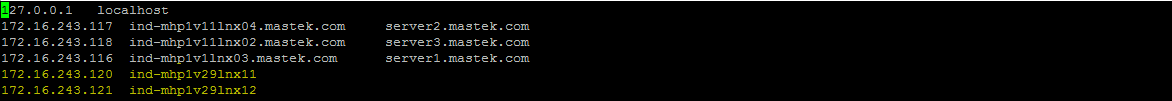
* Servers with MapR Cluster running. In our case, we have setup a 2 node cluster on 172.16.243.120 and 172.16.243.121
* Client on Linux. (We will be using a CentOS but it will work with other distribution as well). In our case, we have a CentOS instance running on 172.16.243.117 server
* Eclipse
* Java 1.8 and above running on the Eclipse machine

## Steps

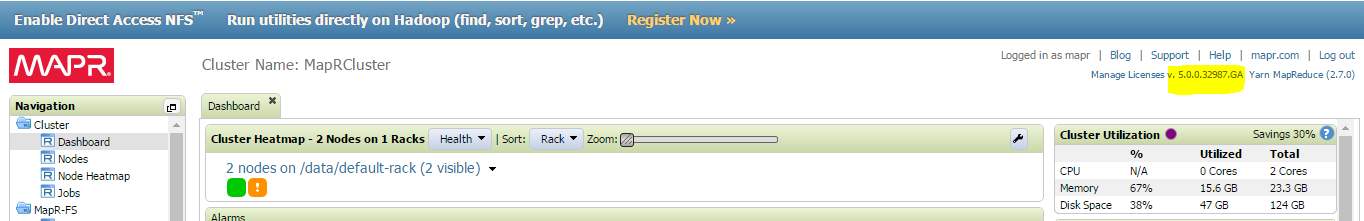
1. Validate if the MapR cluster is up and running. In our case, we setup a 2 node cluster and the same was setup on machines 172.16.243.120 and 172.16.243.121. Visit the MCS (MapR Control System) site to ensure that cluster is up and running. In our case, we validated the same using https://172.16.243.120:8443/



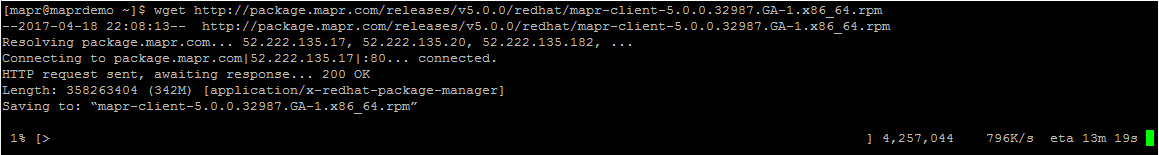
1. In order to access the MapRDB from a remote machine, we need to install mapr client package on it. Also it should be ensured that the remote machine is not part of any Mapr cluster. Steps 3 through 11 will explain steps to setup the remote machine for accessing MapRDB.
2. First, we need to add all the IP address and host names of the MapR Cluster in the hosts file of the remote machine. Login to the remote client machine and edit the /etc/hosts file and add the entries as shown below highlighted in yellow:



1. Save and close the above hosts file.
2. Now we need to download the mapr client package. First we need to identify the version of the MapR Cluster. For this open the MCS site as shown in 1st step and refer to the top right corner as shown below to identify the MapR Cluster version. In our case, the version is **5.0.0.32987.GA**

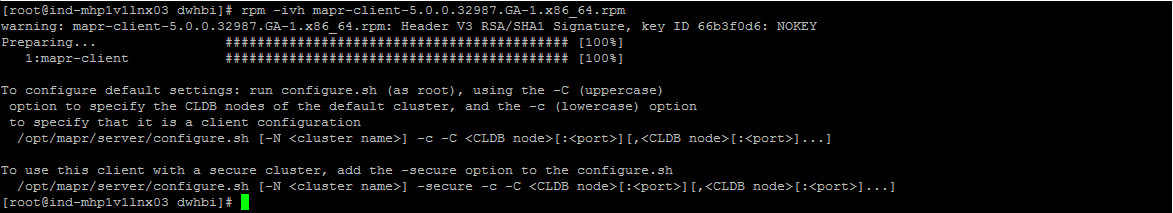


1. Visit the site <http://package.mapr.com/releases/> and click on the version folder that maps to that maps to the version of the MapR Cluster . In our case , the version is 5.0.0 and hence we will go to the site <http://package.mapr.com/releases/v5.0.0/> . Now go into the folder for your appropriate operation system. In our case, the operating system is CentOS and hence we will go to the site <http://package.mapr.com/releases/v5.0.0/redhat/> . You will see a lot of rpm files and download the one that starts with mapr-client-5.0.0.\* as shown below:



1. Once the package is downloaded, switch to root user and run the command from the location where the file is downloaded:

***rpm –ivh mapr-client-5.0.0.32987.GA-1.x86\_64.rpm***

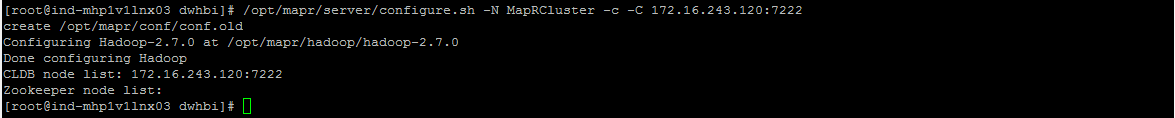


1. The above step will setup the mapr client. Next, we need to configure the client to point it to the cluster. In order to configure the same, we need to know below 4 points:

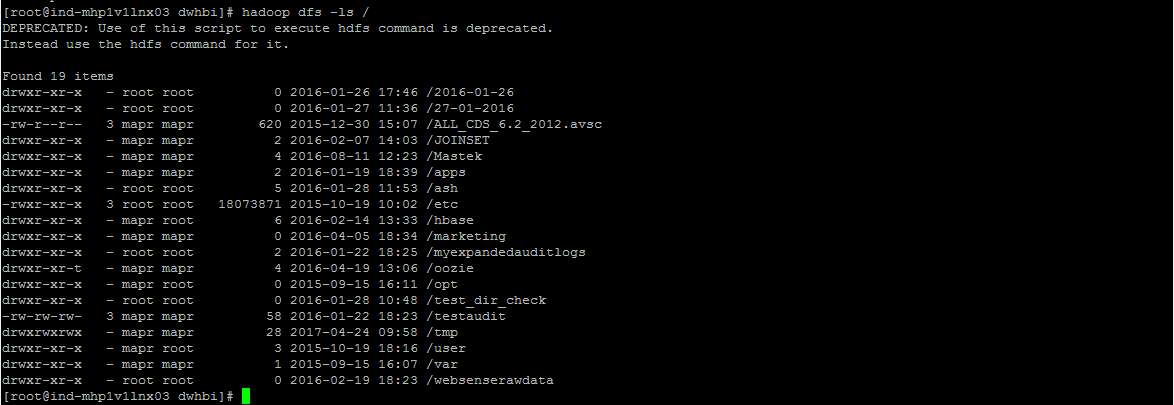
* Name of the MapR Cluster (This can be found out from MCS site. It appears at the top. In our case, the name is MapRCluster).
* If the MapR cluster is secured or not (This can be obtained from administrator. In our case, it’s not secured).
* Servers on which the CLDB service is running (This can be obtained from administrator. In our case, the CLDB service is running on 172.16.243.120 server).

1. Still logged in as root user, we would run the below command to configure the mapr client:

/opt/mapr/server/configure.sh -N MapRCluster -c -C 172.16.243.120:7222



1. If it’s a secure cluster, add the –secure parameter in the above command.
2. In order to test if the mapr client has been successfully installed, just run a basic hadoop command from the command prompt to see if it works before proceeding:



1. Now we will get started with writing a Java Client. Start Eclipse and create an empty Maven project.
2. Open the pom.xml and add the below dependency. The below dependency is as at the time of writing this article. It can change over course of time and hence check appropriately on the maven site.

***<dependency>***

***<groupId>org.apache.hbase</groupId>***

***<artifactId>hbase-client</artifactId>***

***<version>1.1.1-mapr-1602-m7-5.2.0</version>***

***</dependency>***

1. The above artefact is not available on the maven and we need to corresponding repository:

***<repositories>***

***<repository>***

***<id>mapr-releases</id>***

***<url>http://repository.mapr.com/maven/</url>***

***<snapshots><enabled>true</enabled></snapshots>***

***<releases><enabled>true</enabled></releases>***

***</repository>***

***</repositories>***

1. Add the below build property to the pom.xml to generate jar built with dependancies:

***<build>***

***<plugins>***

***<!-- any other plugins -->***

***<plugin>***

***<artifactId>maven-assembly-plugin</artifactId>***

***<executions>***

***<execution>***

***<phase>package</phase>***

***<goals>***

***<goal>single</goal>***

***</goals>***

***</execution>***

***</executions>***

***<configuration>***

***<descriptorRefs>***

***<descriptorRef>jar-with-dependencies</descriptorRef>***

***</descriptorRefs>***

***</configuration>***

***</plugin>***

***</plugins>***

***</build>***

1. In event you face any issue with the pom.xml file throwing error related to tools .jar file, add the below dependency:

***<dependency>***

***<groupId>jdk.tools</groupId>***

***<artifactId>jdk.tools</artifactId>***

***<version><<Replace with Java Version>></version>***

***<scope>system</scope>***

***<systemPath><<Replace with JAVA\_HOME>>/lib/tools.jar</systemPath>***

***</dependency>***

1. You can copy the below import statements in the class file:

**import** java.io.IOException;

**import** org.apache.hadoop.conf.Configuration;

**import** org.apache.hadoop.hbase.HBaseConfiguration;

**import** org.apache.hadoop.hbase.HColumnDescriptor;

**import** org.apache.hadoop.hbase.HTableDescriptor;

**import** org.apache.hadoop.hbase.TableName;

**import** org.apache.hadoop.hbase.client.Get;

**import** org.apache.hadoop.hbase.client.HBaseAdmin;

**import** org.apache.hadoop.hbase.client.HTable;

**import** org.apache.hadoop.hbase.client.Put;

**import** org.apache.hadoop.hbase.client.Result;

**import** org.apache.hadoop.hbase.client.ResultScanner;

**import** org.apache.hadoop.hbase.client.Scan;

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

1. Before we proceed to see code for each of the operation, we need to create a configuration object for HBase and a connection object to the MapRDB table:

//Create an instance of HBase Configuration

Configuration config = HBaseConfiguration.create();

//Create an instance to the MapRDB table.

HTable table = new HTable(config, <<Name of the table with absolute path>>);

1. The below code will help in creating the table:

HBaseAdmin admin = **new** HBaseAdmin(config);

HTableDescriptor tableDescriptor = **new** HTableDescriptor(TableName.*valueOf*(<<Name of the table with absolute path>>));

tableDescriptor.addFamily(**new** HColumnDescriptor(<<column family 1>>));

tableDescriptor.addFamily(**new** HColumnDescriptor(<<column family 2>>));

tableDescriptor.addFamily(**new** HColumnDescriptor(<<column family n>>));

admin.createTable(tableDescriptor);

1. The below code will help in putting a row into the table:

Put p = new Put(Bytes.toBytes((<<Row Key>>));

p.add(Bytes.toBytes((<<column family>>), Bytes.toBytes((<<column name>>),Bytes.toBytes((<<value>>));

table.put(p);

1. The below code will help in getting a row from the table:

Get g = new Get(Bytes.toBytes(<<Row Key>>));

Result r = table.get(g);

byte[] value = r.getValue(Bytes.toBytes(<<column family>>),Bytes.toBytes(<<column name>>));

String valueStr = Bytes.toString(<<value>>);

System.out.println("GET: " + valueStr);

1. The below code will help in deleting the row from the table:

Delete delete = new Delete(Bytes.toBytes(<<Row Key>>));

table.delete(delete);

table.close();

**Other References:**

***Set a timestamp for row deletes.***

delete.setTimestamp(1);

***Delete a specific version in 1 column.***

delete.deleteColumn(Bytes.toBytes("colfam1"), Bytes.toBytes("qual1"), 1);

***Delete all versions in 1 column.***

delete.deleteColumns(Bytes.toBytes("colfam2"), Bytes.toBytes("qual1"));

***Delete the given and all older versions in one column.***

delete.deleteColumns(Bytes.toBytes("colfam2"), Bytes.toBytes("qual3"), 15);

***Delete the entire columnfamily, all columns and version.***

delete.deleteFamily(Bytes.toBytes("colfam3"));

***Delete the given and all older versions in the entire column family , that is, from all columns there in.***

delete.deleteFamily(Bytes.toBytes("colfam3"), 3);

1. The below code will help in scanning the table for multiple rows:

Scan s = new Scan();

s.addColumn(Bytes.toBytes(<<column family>>), Bytes.toBytes(<<column name>>));

ResultScanner scanner = table.getScanner(s);

try {

for (Result rr = scanner.next(); rr != null; rr = scanner.next()) {

System.out.println("Found row: " + rr);

}

} finally {

scanner.close();

}

1. Finally, the jar can be prepared using the maven clean and maven install steps.
2. The generated jar file then can be moved to the remote client machine and executed to perform the above table operations:

java -cp <<MapRDGeneratedJar.jar>> <<ClassFile>>

1. The code can be found on the below github repository:

https://github.com/jainsourabh2/JavaMapRDB

# Hands-On Code Walk Through – C

# References

<https://mapr.com/blog/hbase-and-mapr-db-designed-distribution-scale-and-speed/>