



# Denodo MapReduce Custom Wrapper

Revision 20170214

## NOTE

This document is confidential and proprietary of **Denodo Technologies**.  
No part of this document may be reproduced in any form by any means without prior written authorization of **Denodo Technologies**.

Copyright © 2017  
Denodo Technologies Proprietary and Confidential

## CONTENTS

<b>1 INTRODUCTION.....</b>	<b>3</b>
<b>2 MAPREDUCE.....</b>	<b>4</b>
<b>3 MAPREDUCE CUSTOM WRAPPER.....</b>	<b>6</b>
<b>3.1 USING MAPREDUCE CUSTOM WRAPPER.....</b>	<b>6</b>
<b>3.2 WHY USING SSH TO EXECUTE MAPREDUCE JOBS?.....</b>	<b>8</b>
<b>3.3 WHEN NOT TO USE MAPREDUCE CUSTOM WRAPPER.....</b>	<b>9</b>
<b>4 DEVELOPING A MAPREDUCE JOB.....</b>	<b>10</b>
<b>5 SECURE CLUSTER WITH KERBEROS.....</b>	<b>11</b>
<b>6 SOFTWARE REQUIREMENTS.....</b>	<b>14</b>
<b>7 TROUBLESHOOTING.....</b>	<b>15</b>

## 1 INTRODUCTION

---

mapreduce-customwrapper is a Virtual DataPort custom wrapper for running **map and reduce operations** using **Hadoop**.

The custom wrapper connects to the Hadoop machine **via SSH**, executes a MapReduce job and reads the results from **HDFS**.

## 2 MAPREDUCE

---

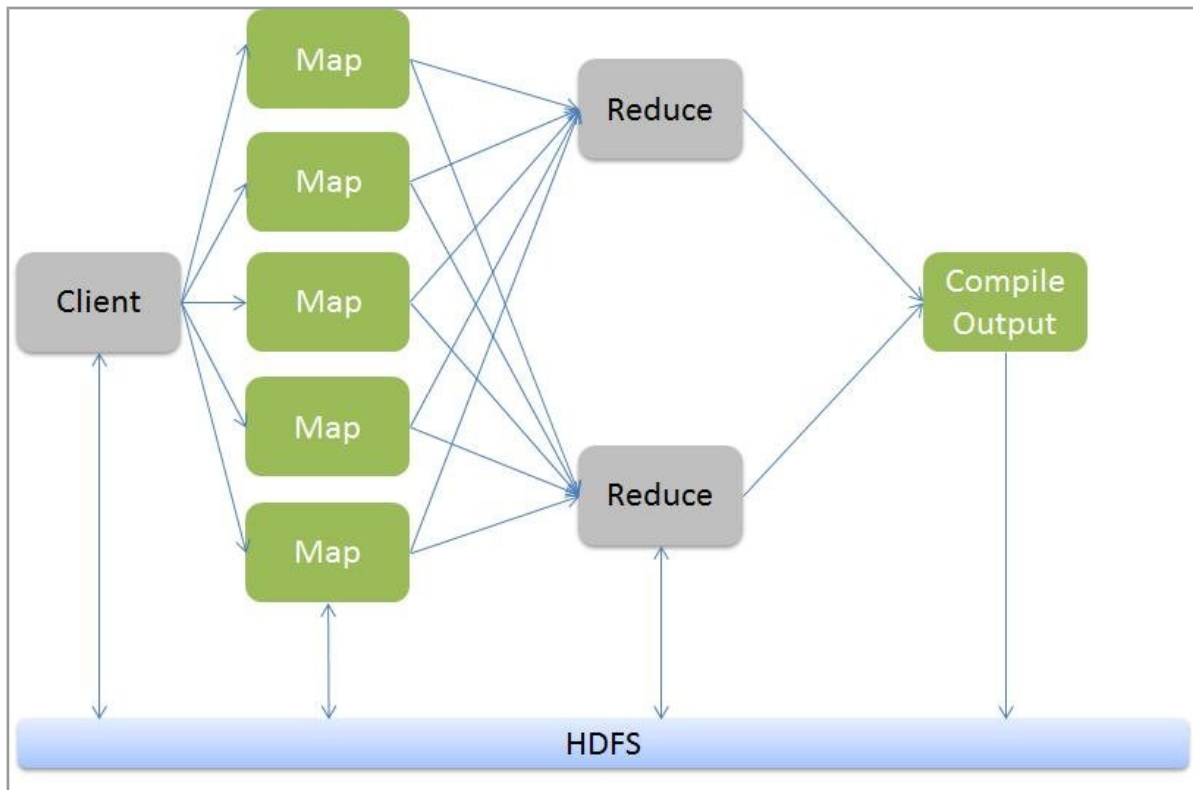
MapReduce is a programming model which uses parallelism to handle **large data sets** in **distributed systems**. This model provides fault tolerance by re-scheduling failed tasks and also considers I/O locality to reduce the network traffic. The process consists of:

1. The master node splits the input file into smaller files and forwards each one to slave nodes.
2. Each slave node operates the **map** function and produces intermediate key-value pairs.
3. The **reduce** function accepts an intermediate key and a set of values for this key and merges these values to get the result.

The model is inspired by the map and reduce functions commonly used in functional programming. Furthermore, the key contribution of the MapReduce framework are not the map and reduce functions, but:

- A simple abstraction for processing any size of data that fits into this programming model. Simple since it hides the details of parallelization, fault-tolerance, locality optimization and load balancing.
- Fault tolerance. The master node monitors workers and restarts chunks of work when necessary.
- Data locality, reducing network overhead.

MapReduce is supported by multiple implementations, being **Apache Hadoop** one of the most popular.



*MapReduce job*

### 3 MAPREDUCE CUSTOM WRAPPER

---

MapReduce custom wrapper connects to the machine where Hadoop is running via SSH and executes a MapReduce job running the following command from the shell:

```
$ hadoop jar <jar> job-parameters
```

When the job finishes the wrapper reads the output stored in an HDFS directory.

#### 3.1 USING MAPREDUCE CUSTOM WRAPPER

Base views created from the MapReduceSSHwrapper need the following **mandatory** parameters:

1. Host IP: Name of the Hadoop machine or its IP address.
2. Host port: Port number to connect to the Hadoop machine, default is 22.
3. User: Username to log into the Hadoop machine.
4. Path to jar in host: Path to the job jar on the Hadoop machine.
5. Output file type: Type of output file generated by the job. You can choose between:
  - AvroFileOutput
  - DelimitedFileOutput
  - MapFileOutput
  - SequenceFileOutput
6. File system URI: A URI whose scheme and authority identify the file system. The scheme determines the file system implementation. The authority is used to determine the host, port, etc.
  - For HDFS the URI has the form `hdfs://<ip>:<port>`.
  - For Amazon S3 the URI has the form `s3n://<id>:<secret>@<bucket>` (Note that since the secret access key can contain slashes, each slash / should be replaced with the string %2F).

Six **optional** parameters:

1. MapReduce job parameter: Arguments required by the MapReduce job. They will be added to the command "hadoop jar...".
2. Custom core-site.xml file: configuration file that overrides the default core parameters (common to HDFS and MapReduce).
3. Custom hdfs-site.xml file: configuration file that overrides the default hdfs parameters.
4. Key class: Key class name implementing `org.apache.hadoop.io.Writable` interface. This parameter is **mandatory** when using `SequenceFileOutput` or `MapFileOutput`.
5. Value class: Value class name implementing `org.apache.hadoop.io.Writable` interface. This parameter is **mandatory**

when using SequenceFileOutput or MapFileOutput.

6. Separator: Delimiter between key/value pairs. This parameter is **mandatory** when using DelimitedFileOutput.

There are also parameters that are **mutually exclusive**:

1. Password: Password associated with the username.
2. Key file: Key file created with PuTTY key file generator.
  - Passphrase: Passphrase associated with the key file.

and

1. Avro schema path: **HDFS path** to the Avro schema file.
2. Avro schema JSON: JSON of the Avro schema.

Only one of these two latter parameters is **mandatory** when using AvroFileOutput.

Host IP	<input type="text" value="melkus.denodo.com"/>
Host port	<input type="text" value="22"/>
User	<input type="text" value="cloudera"/>
Password	<input type="password" value="•••••"/>
Key file	<input type="text" value="None"/>
Passphrase	<input type="text"/>
Path to jar in host	<input type="text" value="/usr/lib/hadoop-mapreduce/hadoop-mapreduce-examples.jar"/>
MapReduce job parameters	<input type="text" value="wordmean /user/cloudera/schema.avsc"/>
Output file type	<input type="text" value="DelimitedFileOutput"/>
File system URI	<input type="text" value="hdfs://melkus.denodo.com:8020"/>
Custom core-site.xml file	<input type="text" value="None"/>
Custom hdfs-site.xml file	<input type="text" value="None"/>
Key class	<input type="text"/>
Value class	<input type="text"/>
Separator	<input type="text"/>
Avro schema path	<input type="text"/>
Avro schema JSON	<input type="text"/>
<input type="checkbox"/> Kerberos enabled	
Kerberos principal name	<input type="text"/>
Kerberos keytab file	<input type="text" value="None"/>
Kerberos password	<input type="text"/>
Kerberos Distribution Center	<input type="text"/>

*MapReduceSSHWrapper base view edition*

View schema:	Field Name	Field Type
	key	text
	value	text

*View schema*

The execution of the view returns the results of the wordmean MapReduce job:

```
SELECT * FROM mapreduce_ds_job
```

Total rows received: 2 (shown 2)	
key	value
count	199
length	881

*View results*

### 3.1.1 Supported values for Key class and Value class

Classes provided for the base view parameters Key class and Value class should implement org.apache.hadoop.io.Writable interface. Value class also supports arrays, i.e. org.apache.hadoop.io.IntWritable[] represents an ArrayWritable of IntWritable.

The Writable classes supported by the custom wrapper are:

- org.apache.hadoop.io.NullWritable
- org.apache.hadoop.io.Text
- org.apache.hadoop.io.IntWritable
- org.apache.hadoop.io.LongWritable
- org.apache.hadoop.io.BooleanWritable
- org.apache.hadoop.io.DoubleWritable
- org.apache.hadoop.io.FloatWritable
- org.apache.hadoop.io.ByteWritable
- An array of any of the previous classes (written as org.apache.hadoop.io.IntWritable[], ...).

## 3.2 WHY USING SSH TO EXECUTE MAPREDUCE JOBS?

The custom wrapper executes a MapReduce job by connecting to the Hadoop machine via SSH and running the command "hadoop jar...". This may seem a complicate way to execute jobs because Hadoop API allows to submit and run jobs programmatically.



When using Hadoop API:

- Job jars have to be added to the Denodo Platform and linked to the data source created for the custom wrapper.
- Base views have to be configured with the job name.

While when using SSH:

- Job jars have to be stored the in the Hadoop machine.
- Base views have to be configured with the path to the job jar in the Hadoop machine.

Consequently, SSH is the preferred way as it avoids Denodo Platform having dependencies with the MapReduce jobs.

### **3.3 WHEN NOT TO USE MAPREDUCE CUSTOM WRAPPER**

MapReduce custom wrapper is synchronous: it invokes a MapReduce job, wait until the job has finished and then reads the output from HDFS. For relatively fast MapReduce jobs synchronous execution would be desirable. However, in situations where a MapReduce job is expected to take a large amount of time asynchronous execution should be preferred.

Denodo Connect provides two components for achieve asynchronous execution of MapReduce jobs:

- SSH custom wrapper: executes a MapReduce job connecting to the Hadoop machine via SSH.
- HDFS custom wrapper: reads the output files of the job stored in HDFS.

## 4 DEVELOPING A MAPREDUCE JOB

---

There are some of the issues to consider when developing a MapReduce job for this custom wrapper:

- A MapReduce job should be packaged in a jar containing at least one main class. The jar does not need to be executable as the name of the main class could be provided as a parameter to the custom wrapper (MapReduce job parameter); this way the jar can contain more than one main class and therefore more than one job.
- The job jar should be located in an accessible path in the machine running Hadoop. The jar is not stored in HDFS but in the machine file system.
- The job receives the parameters in the following order: first those indicated in the property MapReduce job parameters, then the file output path that is established by the custom wrapper.
- The output folder is named as “denodo\_output\_X”, where x is a random value.
- The job output folder is deleted after the custom wrapper execution.

Below there is the skeleton of the main class of a MapReduce job.

```
public class MapReduceDriverExample {

    public static void main(String[] args) {

        String input = args[0];
        String output = args[1];

        Job job = new Job();
        job.setJarByClass(MapReduceDriverExample.class);
        job.setJobName("example-" + System.nanoTime());

        // Input and output are in HDFS
        FileInputFormat.setInputPaths(conf, new Path(input));
        FileOutputFormat.setOutputPath(conf, new Path(output));

        job.setMapperClass(Mapper1.class);
        job.setReducerClass(Reducer1.class);
        job.setOutputFormat(SequenceFileOutputFormat.class);

        job.setMapOutputKeyClass(Text.class);
        job.setMapOutputValueClass(IntWritable.class);
        job.setOutputKeyClass(Text.class);
        job.setOutputValueClass(IntWritable.class);

        boolean success = job.waitForCompletion(true);

        System.exit(success ? 0 : 1);
    }
}
```

*MapReduce job main class skeleton*

## 5 SECURE CLUSTER WITH KERBEROS

---

The configuration required for accessing a Hadoop cluster with Kerberos enabled is the same as the one needed to execute MapReduce jobs and, additionally, the user must supplied the Kerberos credentials.

The Kerberos parameters are:

- Kerberos enabled: Check it when accessing a Hadoop cluster with Kerberos enabled (required).
- Kerberos principal name: Kerberos v5 Principal name to access HDFS, e.g. primary/instance@realm (optional).
- Kerberos keytab file: Keytab file containing the key of the Kerberos principal (optional).
- Kerberos password: Password associated with the principal (optional).
- Kerberos Distribution Center: Kerberos Key Distribution Center (optional).

hdfs-mapreducewrapper provides **three ways** for accessing a kerberized Hadoop cluster:

1. The client has a valid Kerberos ticket in the **ticket cache** obtained, for example, using the kinit command in the Kerberos Client.  
In this case only the Kerberos enabled parameter should be checked. The MapReduce wrapper would use the Kerberos ticket to authenticate itself against the Hadoop cluster.
2. The client does not have a valid Kerberos ticket in the ticket cache. In this case you should provide the Kerberos principal name parameter and
  - 2.1. Kerberos keytab file parameter or
  - 2.2. Kerberos password parameter

In all these **three scenarios** the **krb5.conf** file should be present in the file system. Below there is an example of the Kerberos configuration file:

```
[libdefaults]
renew_lifetime = 7d
forwardable = true
default_realm = EXAMPLE.COM
ticket_lifetime = 24h
dns_lookup_realm = false
dns_lookup_kdc = false

[domain_realm]
sandbox.hortonworks.com = EXAMPLE.COM
cloudera = CLOUDERA
```

```
[realms]
EXAMPLE.COM = {
    admin_server = sandbox.hortonworks.com
    kdc = sandbox.hortonworks.com
}






CLOUDERA = {
    kdc = quickstart.cloudera
    admin_server = quickstart.cloudera
    max_renewable_life = 7d 0h 0m 0s
    default_principal_flags = +renewable
}

[logging]
default = FILE:/var/log/krb5kdc.log
admin_server = FILE:/var/log/kadmind.log
kdc = FILE:/var/log/krb5kdc.log
```

The algorithm to locate the `krb5.conf` file is the following:

- If the system property `java.security.krb5.conf` is set, its value is assumed to specify the path and file name.
- If that system property value is not set, then the configuration file is looked for in the directory
  - `<java-home>\lib\security` (Windows)
  - `<java-home>/lib/security` (Solaris and Linux)
- If the file is still not found, then an attempt is made to locate it as follows:
  - `/etc/krb5/krb5.conf` (Solaris)
  - `c:\winnt\krb5.ini` (Windows)
  - `/etc/krb5.conf` (Linux)

There is an **exception**. If you are planning to create MapReduce views that use the **same Key Distribution Center and the same realm** the Kerberos Distribution Center parameter can be provided instead of having the `krb5.conf` file in the file system.

Host IP	<input type="text" value="quickstart.cloudera"/>
Host port	<input type="text" value="22"/>
User	<input type="text" value="cloudera"/>
Password	<input type="password" value="•••••"/>
Key file	<input type="text" value="None"/> 
Passphrase	<input type="text"/>
Path to jar in host	<input type="text" value="/usr/lib/hadoop-mapreduce/hadoop-mapreduce-examples.jar"/>
MapReduce job parameters	<input type="text" value="wordmean /user/cloudera/file.avro"/>
Output file type	<input type="text" value="DelimitedFileOutput"/> 
File system URI	<input type="text" value="hdfs://quickstart.cloudera"/>
Custom core-site.xml file	<input type="text" value="None"/> 
Custom hdfs-site.xml file	<input type="text" value="Local"/> 
	<input type="text" value="C:/Work/hdfs-site.xml"/>
Key class	<input type="text"/>
Value class	<input type="text"/>
Separator	<input type="text"/>
Avro schema path	<input type="text"/>
Avro schema JSON	<input type="text"/>
	<input checked="" type="checkbox"/> Kerberos enabled
Kerberos principal name	<input type="text" value="cloudera-scm/admin\@CLOUDERA"/>
Kerberos keytab file	<input type="text" value="None"/> 
Kerberos password	<input type="password" value="•••••"/>
Kerberos Distribution Center	<input type="text" value="quickstart.cloudera"/>

[View edition](#)

**Note:** As the wrapper connects to the cluster via SSH and executes the MapReduce job with the command `hadoop jar`, the local user account must have a Kerberos valid ticket before running the MapReduce wrapper. Tickets can be requested with the `kinit` command.

## 6 SOFTWARE REQUIREMENTS

---

mapreduce-customwrapper has been tested in **Cloudera** QuickStart VM 5.8 and in **Hortonworks** Sandbox 2.5 using Hadoop v2.7.3 and Avro v1.8.1.

## 7 TROUBLESHOOTING

---

### Symptom

Error message: "Host Details : local host is: "<your domain/your IP>"; destination host is: "<hadoop IP>:hadoop port>".

### Resolution

It is a version mismatch problem. Hadoop server version is **older** than the version distributed in the custom wrapper artifact denodo-mapreduce-customwrapper-5.0-xxx-jar-with-dependencies, which is Hadoop v2.4.0.

To solve the problem you should use the custom wrapper artifact denodo-mapreduce-customwrapper-5.0-xxx and copy the Hadoop server libraries to the \$DENODO\_PLATFORM\_HOME/extensions/thirdparty/lib directory.

### Symptom

Error message: "Server IPC version X cannot communicate with client version Y".

### Resolution

It is a version mismatch problem. Hadoop server version is **newer** than the version distributed in the custom wrapper artifact denodo-mapreduce-customwrapper-5.0-xxx-jar-with-dependencies, which is Hadoop v2.4.0.

To solve the problem you should use the custom wrapper artifact denodo-mapreduce-customwrapper-5.0-xxx and copy the Hadoop server libraries to the \$DENODO\_PLATFORM\_HOME/extensions/thirdparty/lib directory.

### Symptom

Error message: "denodo\_output\_XXX/\_logs is not a file".

### Resolution

Job logs refers to the events and configuration for a completed job. They are retained to provide interesting information for the user running a job. These job history files are stored on the local filesystem of the jobtracker in a history subdirectory of the logs directory.

A second copy is also stored for the user in the \_logs/history subdirectory of the job's output directory. This is the directory the custom wrapper is complaining about because it expects only output files.

By setting to none the property `hadoop.job.history.user.location` in the `mapred-site.xml` config file or in the mapreduce job configuration no user job history is saved in the job's output directory.

## Symptom

Error message: "No valid credentials provided (Mechanism level: Failed to find any Kerberos tgt)".

## Resolution

As the wrapper connects to the cluster via SSH and executes the MapReduce job with the command `hadoop jar`, the local user account must have a Kerberos valid ticket before running the MapReduce wrapper. Tickets can be requested with the `kinit` command.

## Symptom

Error message: "SIMPLE authentication is not enabled. Available:[TOKEN, KERBEROS]".

## Resolution

You are trying to connect to a Kerberos-enabled Hadoop cluster. You should configure the custom wrapper accordingly. See [Secure cluster with Kerberos section](#) for **configuring Kerberos** on this custom wrapper.

## Symptom

Error message: "Cannot get Kerberos service ticket: KrbException: Server not found in Kerberos database (7) ".

## Resolution

Check that `nslookup` is returning the fully qualified hostname of the KDC. If not, modify the `/etc/hosts` of the client machine for the KDC entry to be of the form "IP address fully.qualified.hostname alias".