



## HDFS Custom Wrapper

### 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 © 2013  
Denodo Technologies Proprietary and Confidential

## CONTENTS

<b>1 INTRODUCTION.....</b>	<b>3</b>
<b>2 HDFS.....</b>	<b>3</b>
2.1 DELIMITED TEXT FILES.....	4
2.2 SEQUENCEFILES.....	4
2.3 MAPFILES.....	6
2.4 MAP FILE FORMAT.....	6
2.5 AVRO FILES.....	6
<b>3 HDFS CUSTOM WRAPPER.....</b>	<b>7</b>
3.1 HDFSDELIMITEDTEXTFILEWRAPPER.....	7
3.2 HDFSSEQUENCEFILEWRAPPER.....	9
3.3 HDFSMAPIFILEWRAPPER.....	10
3.4 HDFSAVROFILEWRAPPER.....	11
3.5 HTTPFSFILEWRAPPER.....	14
<b>4 HDFS COMPRESSED FILES.....</b>	<b>16</b>
<b>6 SOFTWARE REQUIREMENTS.....</b>	<b>17</b>

## 1 INTRODUCTION

---

`hdfs-customwrapper` library contains five Virtual DataPort custom wrappers capable of reading several file formats stored in **Hadoop Distributed File System** (HDFS).

Supported formats are:

- Delimited text files
- Sequence files
- Map files
- Avro files

## 2 HDFS

---

HDFS is a distributed, scalable, and portable file system written in Java for the [Hadoop framework](#).

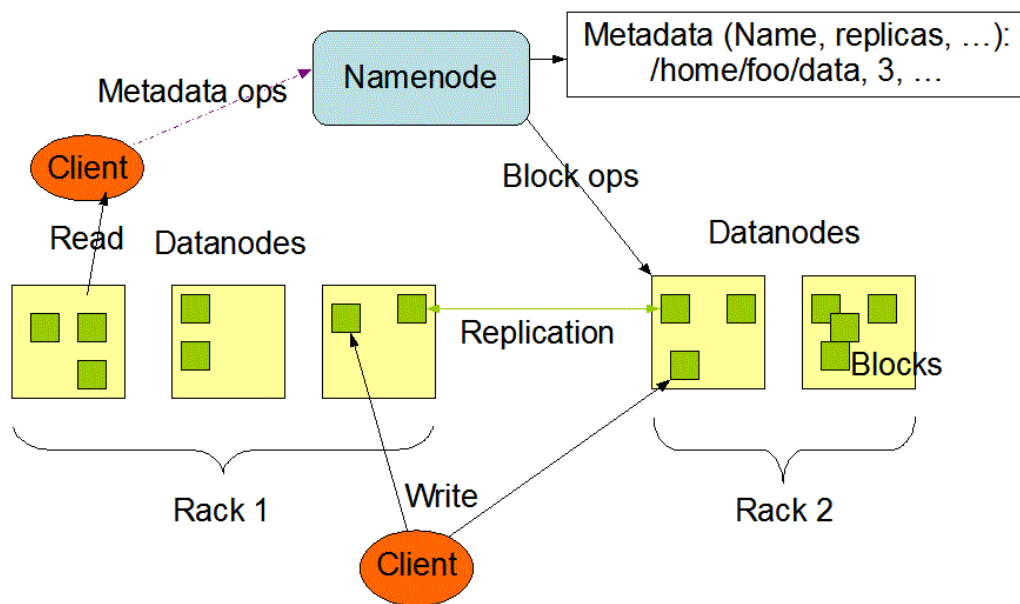
In HDFS data is divided into blocks and copies of these blocks are stored on other servers in the Hadoop cluster. That is, an individual file is actually stored as smaller blocks that are replicated across multiple servers in the entire cluster. This redundancy offers multiple benefits for Big Data processing:

- Higher availability.
- Better scalability: map and reduce functions can be executed on smaller subsets of large data sets.
- Data locality: move the computation closer to the data to reduce latency.

HDFS has a master/slave architecture in which the **NameNode**, the master, manages the file system namespace and regulates clients access to files. And the **DataNodes**, the slaves, manage storage attached to the nodes that they run on.

The NameNode executes file system namespace operations like opening, closing, and renaming files and directories. It also determines the mapping of blocks to

DataNodes. The DataNodes are responsible for serving read and write requests from the file system's clients. The DataNodes also perform block creation, deletion, and replication upon instruction from the NameNode.



*HDFS Architecture*

## 2.1 DELIMITED TEXT FILES

Delimited text files store plain text key/value pairs. The key/value pairs are delimited by a separator such as tab, space, comma, etc.

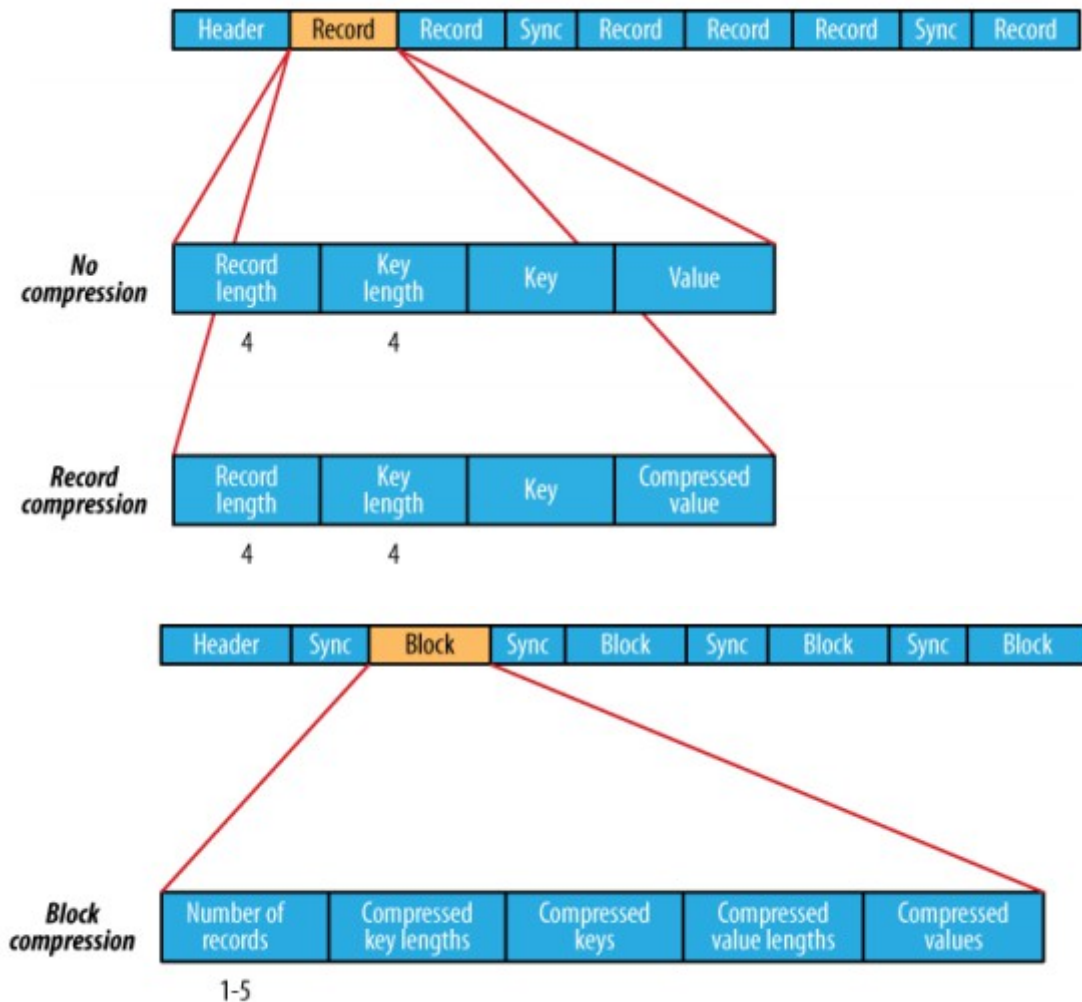
## 2.2 SEQUENCEFILES

Sequence files are binary record-oriented files, where each record has a serialized key and a serialized value.

The Hadoop framework supports compressing and decompressing sequence files transparently. Therefore, sequence files has three available formats:

- No compression.

- Record compression: only values are compressed.
- Block compression: both keys and values are collected in 'blocks' separately and compressed.



### Sequence file formats

The three formats share a header that contains information which allows the reader to recognize their format:

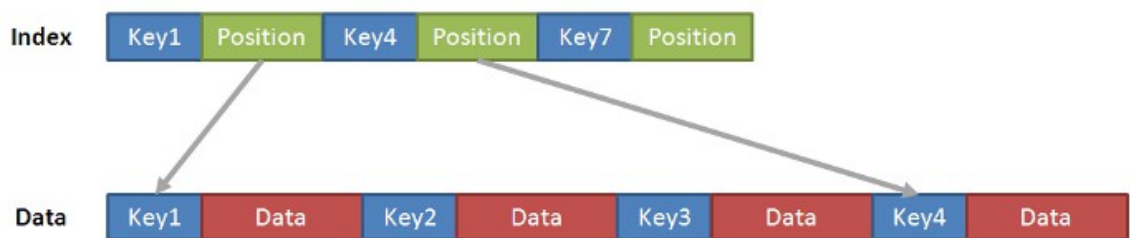
- Version: 3 bytes of magic header SEQ, followed by 1 byte of actual version number.
- Key class name.
- Value class name.
- Compression: specifies if compression is turned on for keys/values.
- Block compression: specifies if block-compression is turned on for keys/values.

- Compression codec: `CompressionCodec` class which is used for compression of keys and/or values (if compression is enabled).
- Metadata.
- Sync: a sync marker to denote end of the header.

## 2.3 MAPFILES

A map file consists of two sequence files. The data file ("/data") is identical to the sequence file and contains the data stored as binary key/value pairs. The index file ("/index"), which contains a key/value map with seek positions inside the data file to quickly access the data.

The index file is populated with the key and a `LongWritable` that contains the starting byte position of the record. Index does not contains all the keys but just a fraction of the keys. The index is read entirely into memory.



## 2.4 MAP FILE FORMAT

## 2.5 AVRO FILES

[Avro](#) is a data serialization format.

Avro data files are self-describing, containing the full schema for the data in the file. An Avro schema is defined using JSON. The schema allows you to define two types of data:

- primitive data types: string, integer, long, float, double, byte, null and boolean.
- complex type definitions: a record, an array, an enum, a map, a union or a fixed type.

```
{ "namespace": "example.avro",
  "type": "record",
  "name": "User",
  "fields": [
    { "name": "name", "type": "string",
      { "name": "favorite_number", "type": ["int",
"null"]},
    { "name": "favorite_color", "type": ["string",
"null"]}
  ]
}
```

### *Avro schema*

Avro relies on schemas. When Avro data is read, the schema used when writing it is always present. This allows each datum to be written with no per-value overheads, making serialization both fast and small. This also facilitates use with dynamic, scripting languages, since data, together with its schema, is fully self-describing.

## 3 HDFS CUSTOM WRAPPER

---

`hdfs-customwrapper` library includes five custom wrappers:

- `com.denodo.connect.hadoop.hdfs.wrapper.HDFSDelimitedTextFileWrapper`
- `com.denodo.connect.hadoop.hdfs.wrapper.HDFSSequenceFileWrapper`
- `com.denodo.connect.hadoop.hdfs.wrapper.HDFSMapFileWrapper`
- `com.denodo.connect.hadoop.hdfs.wrapper.HDFSAvroFileWrapper`
- `com.denodo.connect.hadoop.hdfs.wrapper.HttpFsFileWrapper`

### 3.1 HDFSDELIMITEDTEXTFILEWRAPPER

Custom wrapper for reading key/value delimited text files stored in HDFS.

The base views created from the `HDFSDelimitedTextFileWrapper` need the following **mandatory** parameters:

- **Host IP:** IP of the NameNode (the HDFS master).
- **Host port:** port of the NameNode. By default the NameNode is listening on port 8020 for filesystem metadata operations.
- **Path:** input path for the file or the directory containing the files.
- **Separator:** delimiter between the keys and values.

admin

- hdfs\_avro\_files
- hdfs\_delimited\_files
- hdfs\_map\_files
- hdfs\_sequence\_files

### Edit parameter values

Host IP	<input type="text" value="192.168.150.128"/>
Host port	<input type="text" value="8020"/>
Path	<input type="text" value="/user/sandbox/delimited/text"/>
Separator	<input type="text" value=" "/>

*HDFSDelimitedTextFileWrapper base view edition*

delimited_file_test	
key	text
value	text

*View schema*

The execution of the wrapper returns the key/value pairs contained in the file or group of files, if the `Path` input parameter denotes a directory.



Execute view delimited_file_test	
Total rows received: 5 (shown 5)	
key	value
1	One, two, buckle my shoe
2	Three, four, shut the door
3	Five, six, pick up sticks
4	Seven, eight, lay them straight
5	Nine, ten, a big fat hen

*View results*

### 3.2 HDFSSEQUENCEFILEWRAPPER

Custom wrapper for reading sequence files stored in HDFS.

The base views created from the `HDFSSequenceFileWrapper` need the following **mandatory** parameters:

- `Host IP`: IP of the NameNode (the HDFS master).
- `Host port`: port of the NameNode. By default the NameNode is listening on port 8020 for filesystem metadata operations.
- `Path`: input path for the file or the directory containing the files.
- `Key class`: key class name implementing `org.apache.hadoop.io.Writable` interface.
- `Value class`: value class name implementing `org.apache.hadoop.io.Writable` interface.

admin

- hdfs\_avro\_files
- hdfs\_delimited\_files
- hdfs\_map\_files
- hdfs\_sequence\_files
- delimited\_file\_test

### Edit parameter values

Host IP	192.168.150.128
Host port	8020
Path	/user/sandbox/sequence/sequence.seq
Key class	org.apache.hadoop.io.IntWritable
Value class	org.apache.hadoop.io.Text

*HDFSSequenceFileWrapper base view edition*

sequence_files_test	
key	int
value	text

*View schema*

The execution of the wrapper returns the key/value pairs contained in the file or group of files, if the `Path` input parameter denotes a directory.

Execute view sequence_files_test	
Total rows received: 100 (shown 100)	
key	value
100	One, two, buckle my shoe
99	Three, four, shut the door
98	Five, six, pick up sticks
97	Seven, eight, lay them straight
96	Nine, ten, a big fat hen
95	One, two, buckle my shoe
94	Three, four, shut the door
93	Five, six, pick up sticks
92	Seven, eight, lay them straight
91	Nine, ten, a big fat hen
90	One, two, buckle my shoe

*View results*

### 3.3 HDFSMAPIWRAPPER

Custom wrapper for reading map files stored in HDFS.

The base views created from the `HDFSMapFileWrapper` need the following **mandatory** parameters:

- `Host IP`: IP of the NameNode (the HDFS master).
- `Host port`: port of the NameNode. By default the NameNode is listening on port 8020 for filesystem metadata operations.
- `Path`: input path for the file or the directory containing the files.
- `Key class`: `key class name` implementing `org.apache.hadoop.io.WritableComparable` interface. `WritableComparable` is used because records are sorted in **key order**.
- `Value class`: `value class name` implementing `org.apache.hadoop.io.Writable` interface.

admin

- hdfs\_avro\_files
- hdfs\_delimited\_files
- hdfs\_map\_files
- hdfs\_sequence\_files
- delimited\_file\_test
- sequence\_files\_test

### Edit parameter values

Host IP	192.168.150.128
Host port	8020
Path	/user/sandbox/map
Key class	org.apache.hadoop.io.IntWritable
Value class	org.apache.hadoop.io.Text

HDFSMapFileWrapper base view edition

map_files_test	
key	int
value	text

View schema

The execution of the wrapper returns the key/value pairs contained in the file or group of files, if the `Path` input parameter denotes a directory.

Execute view map_files_test	
Total rows received: 1024 (shown 150)	
key	value
1	One, two, buckle my shoe
2	Three, four, shut the door
3	Five, six, pick up sticks
4	Seven, eight, lay them straight
5	Nine, ten, a big fat hen
6	One, two, buckle my shoe
7	Three, four, shut the door
8	Five, six, pick up sticks
9	Seven, eight, lay them straight
10	Nine, ten, a big fat hen
11	One, two, buckle my shoe

View results

### 3.4 HDFS AVRO FILE WRAPPER

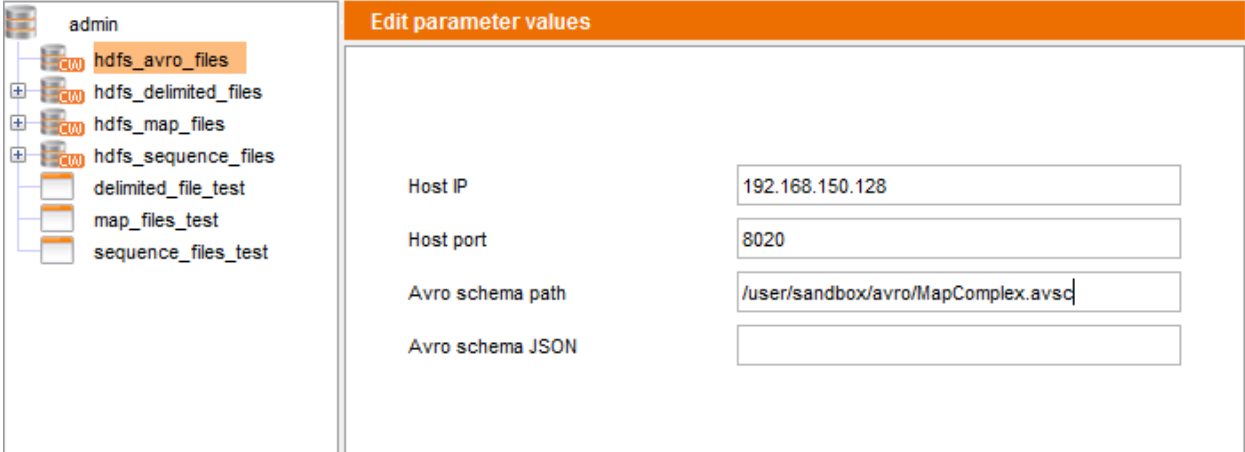
Custom wrapper for reading Avro files stored in HDFS.

The base views created from the `HDFSAvroFileWrapper` need the following **mandatory** parameters:

- `Host IP`: IP of the NameNode (the HDFS master).
- `Host port`: port of the NameNode. By default the NameNode is listening on port 8020 for filesystem metadata operations.

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

- `Avro schema path`: input path for the Avro schema file.
- `Avro schema JSON`: JSON of the Avro schema.



Edit parameter values	
Host IP	192.168.150.128
Host port	8020
Avro schema path	/user/sandbox/avro/MapComplex.avsc
Avro schema JSON	

*HDFSAvroFileWrapper base view edition*

```
{
  "type": "map",
  "values": {
    "type": "record",
    "name": "ATM",
    "fields": [
      { "name": "serial_no", "type": "string" },
      { "name": "location", "type": { "type": "map", "values": { "type": "array", "items": "int" } } }
    ]
  }
}
```

*Avro schema*

avro_files_test	
avrofilepath	text
map	hdfs_avro_files_map
key	text
atm	hdfs_avro_files_map_map_record_atm
serial_no	text
location	hdfs_avro_files_map_map_record_atm_location
key	text
value	hdfs_avro_files_map_map_record_atm_location_location_record_value
int	int

*View schema*

The execution of the wrapper returns the values contained in the Avro file specified in the `WHERE` clause.

Execute view avro_files_test	
Total rows received: 1 (shown 1)	
avrofilepath	map
/user/sandbox/avro/MapComplex.avro	[Array]...



**RESULT -> map**

key	atm
atm1	[Register]...
atm4	[Register]...
atm2	[Register]...

**RESULT -> map -> atm**

serial_no	location
zxy555	[Array]...



RESULT -> map -> atm -> location

key	value
central	[Array]...
downtown	[Array]...



RESULT -> map -> atm -> location -> value

int
1
2
3

*View results*

### 3.5 HTTPFSFILEWRAPPER

Custom wrapper for reading key/value delimited text files stored in HDFS using HttpFS.

#### 3.5.1 About HttpFS

[Apache Hadoop HttpFS](#) is a service that provides HTTP access to HDFS.

HttpFS has a REST HTTP API supporting all HDFS File System operations, both read and write.

The advantage of HttpFS are:

- **Version-independent** REST-based protocol which means that can be read and written to/from Hadoop clusters no matter their version. This addresses the issue of using the Java API (RPC-based) that requires both the client and the Hadoop cluster to share the same version. Upgrading one without the other causes serialization errors meaning the client cannot interact with the cluster.
- Read and write data in HDFS in a cluster behind a firewall. The HttpFS server acts as a gateway and is the only system that is allowed to send and receive data through the firewall.

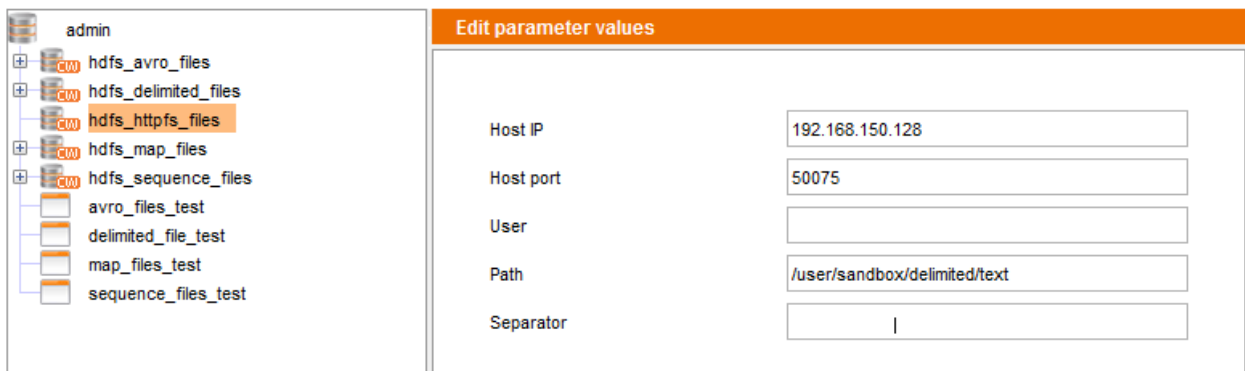
#### 3.5.2 Custom wrapper

The base views created from the `HttpFsFileWrapper` need the following **mandatory** parameters:

- `Host IP`: IP of the DataNode.
- `Host port`: port of the DataNode. By default the DataNode is listening on port 50075 for WebUI access.
- `Path`: input path for the file.
- `Separator`: delimiter between the keys and values.

Optional parameter:

- `User`: The name of the the authenticated user when security is off. If is not set, the server may either set the authenticated user to a default web user, if there is any, or return an error response.



*HttpFsFileWrapper base view edition*

httpfs_files_test	
key	text
value	text

*View schema*

The execution of the wrapper returns the key/value pairs contained in the file.

Execute view httpfs_files_test	
Total rows received: 5 (shown 5)	
key	value
1	One, two, buckle my shoe
2	Three, four, shut the door
3	Five, six, pick up sticks
4	Seven, eight, lay them straight
5	Nine, ten, a big fat hen

*View results*

## 4 HDFS COMPRESSED FILES

---

Hadoop is intended for storing large data volumes, so compression becomes a mandatory requirement here. There are different compression formats available like DEFLATE, GZip, BZip2, Snappy and LZO.

Hadoop has native implementations of compression libraries for performance reasons and for non-availability of Java implementations:

Compression format	Java implementation	Native implementation
DEFLATE	Yes	Yes
gzip	Yes	Yes
bzip2	Yes	No
LZO	No	Yes
Snappy	No	Yes

Compression library implementations

For reading HDFS compressed files using the `hdfs-customwrapper` library there are two options:

- Use the Java implementation.  
`hdfs-customwrapper` handles compressed files transparently.
- Use the native implementation (for performance reasons or for non-availability of Java implementation).  
`hdfs-customwrapper` must have Hadoop native libraries in the `java.library.path`.

Hadoop comes with prebuilt native compression libraries for 32- and 64-bit Linux, which could be found in the `lib/native` directory. For other platforms, the libraries must be compiled, following the instructions on the Hadoop wiki at <http://wiki.apache.org/hadoop/NativeHadoop>.



In the HortonWork Sandbox (see *Software requirements* section) native libraries could be found at `/usr/lib/hadoop/lib/native/{os.arch}`.

## 5

## 6 SOFTWARE REQUIREMENTS

---

hdfs-customwrapper has been tested in the HortonWork Data Platform 1.2 Sandbox using Hadoop v1.1.2 and Avro v1.7.2.

**HortonWork Sandbox** is a complete, self contained virtual machine with Apache Hadoop pre-configured.

The VMWare image of the Sandbox can be downloaded from:  
<https://s3.amazonaws.com/hw-sandbox/Bimota/Hortonworks+Sandbox+1.2+1-21-2012-1+vmware.ova>

The VirtualBoximage of the Sandbox can be downloaded from:  
<https://s3.amazonaws.com/hw-sandbox/Bimota/Hortonworks+Sandbox+1.2+1-21-2012-1+VirtualBox.ova>