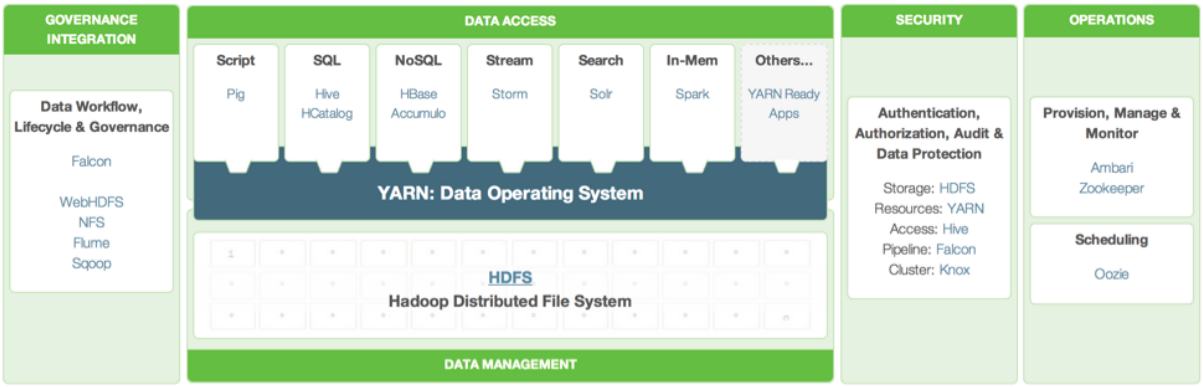
Introducing Apache Hadoop to Java Developers

Apache Hadoop is a community driven open-source project goverened by the Apache Software Foundation .

It was originally implemented at Yahoo based on papers published by Google in 2003 and 2004. Hadoop committers today work at several different organizations like Hortonworks, Microsoft, Facebook, Cloudera and many others around the world.

Since then Apache Hadoop has matured and developed to become a data platform for not just processing humongous amount of data in batch but with the advent of YARN it now supports many diverse workloads such as Interactive queries over large data with Hive on Tez, Realtime data processing with Apache Storm, super scalable NoSQL datastore like HBase, in-memory datastore like Spark and the list goes on.



For this introductory tutorial for Hadoop Developers we are going to focus on the basics, like:

- The core concepts of Apache Hadoop
- Writing a MapReduce program

We have many tutorials which you can use with the Hortonworks Sandbox to learn about a rich and diverse set of components of the Hadoop platform.

Core of Apache Hadoop

- The Hadoop Distributed File System (HDFS)
- MapReduce

A set of machines running HDFS and MapReduce is known as a Hadoop Cluster. Individual machines are known as nodes. A cluster can have as few as one node to as many as several thousands. For most application scenarios Hadoop is linearly scalable, which means you can expect better performance by simply adding more nodes.

MapReduce

MapReduce is a method for distributing a task across multiple nodes. Each node processes data stored on that node to the extent possible.

A running Map Reduce job consists of various phases such as `Map -> Sort -> Shuffle -> Reduce`

The primary advantages of abstracting your jobs as MapReduce running over a distributed infrastructure like CPU and Storage are:

- Automatic parallelization and distribution of data in blocks across a distributed, scale-out infrastructure.
- Fault-tolerance against failure of storage, compute and network infrastructure
- Deployment, monitoring and security capability
- A clean abstraction for programmers

Most MapReduce programs are written in Java. It can also be written in any scripting language using the Streaming API of Hadoop. MapReduce abstracts all the low level plumbing away from the developer such that developers can concentrate on writing the Map and Reduce functions.

The MapReduce Concepts and Terminology

MapReduce jobs are controlled by a software daemon known as the `JobTracker`. The JobTracker resides on a 'master node'. Clients submit MapReduce jobs to the JobTracker. The JobTracker assigns Map and Reduce tasks to other nodes on the cluster.

These nodes each run a software daemon known as the `TaskTracker`. The TaskTracker is responsible for actually instantiating the Map or Reduce task, and reporting progress back to the JobTracker

A `job` is a program with the ability of complete execution of Mappers and Reducers over a dataset. A `task` is the execution of a single Mapper or Reducer over a slice of data.

There will be at least as many task attempts as there are tasks. If a task attempt fails, another will be started by the JobTracker. Speculative execution can also result in more task attempts than completed tasks.

MapReduce: The Mapper

Hadoop attempts to ensure that Mappers run on nodes which hold their portion of the data locally, to minimize network traffic. Multiple Mappers run in parallel, each processing a portion of the input data.

The Mapper reads data in the form of key/value pairs. It outputs zero or more key/value pairs

```
map(in_key, in_value) -> (inter_key, inter_value) list
```

The Mapper may use or completely ignore the input key. For example, a standard pattern is to read a line of a file at a time. The key is the byte offset into the file at which the line starts. The value is the contents of the line itself. Typically the key is considered irrelevant. If the Mapper writes anything out, the output must be in the form of key/value pairs.

MapReduce: The Reducer

After the Map phase is over, all the intermediate values for a given intermediate key are combined together into a list. This list is given to a Reducer. There may be a single Reducer, or multiple Reducers, this is specified as part of

the job configuration. All values associated with a particular intermediate key are guaranteed to go to the same Reducer.

The intermediate keys, and their value lists, are passed to the Reducer in sorted key order. This step is known as the 'shuffle and sort'. The Reducer outputs zero or more final key/value pairs. These are written to HDFS. In practice, the Reducer usually emits a single key/value pair for each input key.

It is possible for some Map tasks to take more time to complete than the others, often due to faulty hardware, or underpowered machines. This might cause a bottleneck as all mappers need to finish before any reducers can kick-off. Hadoop uses speculative execution to mitigate against such situations. If a Mapper appears to be running sluggishly than the others, a new instance of the Mapper will be started on another machine, operating on the same data. The results of the first Mapper to finish will be used. Hadoop will kill off the Mapper which is still running.

Writing a MapReduce Program

In this section you will learn how to use the Hadoop API to write a MapReduce program in Java

Each of the portions (RecordReader, Mapper, Partitioner, Reducer, etc.) can be created by the developer. The developer is expected to atleast write the Mapper, Reducer, and driver code.

The MapReduce Example

WordCount example reads text files and counts how often words occur. The input is text files and the output is text files, each line of which contains a word and the count of how often it occured, separated by a tab.

Each mapper takes a line as input and breaks it into words. It then emits a key/value pair of the word and 1. Each reducer sums the counts for each word and emits a single key/value with the word and sum.

As an optimization, the reducer is also used as a combiner on the map outputs. This reduces the amount of data sent across the network by combining each word into a single record.

To run the example, the command syntax is

```
hadoop jar hadoop-*-examples.jar wordcount [-m <#maps>] [-r <#reducers>] <in-dir>
<out-dir>
```

All of the files in the input directory are read and the counts of words in the input are written to the output directory. It is assumed that both inputs and outputs are stored in HDFS. If your input is not already in HDFS, but is rather in a local file system somewhere, you need to copy the data into HDFS using a command like this:

```
hadoop dfs -copyFromLocal <local-dir> <hdfs-dir>
```

Below is the standard wordcount example implemented in Java:

```
    package org.myorg;

    import java.io.IOException;
    import java.util.*;

    import org.apache.hadoop.fs.Path;
    import org.apache.hadoop.conf.*;
    import org.apache.hadoop.io.*;
    import org.apache.hadoop.mapreduce.*;
    import org.apache.hadoop.mapreduce.lib.input.FileInputFormat;
    import org.apache.hadoop.mapreduce.lib.input.TextInputFormat;
    import org.apache.hadoop.mapreduce.lib.output.FileOutputFormat;
    import org.apache.hadoop.mapreduce.lib.output.TextOutputFormat;

    public class WordCount {

     public static class Map extends Mapper<LongWritable, Text, Text, IntWritable> {
        private final static IntWritable one = new IntWritable(1);
        private Text word = new Text();

        public void map(LongWritable key, Text value, Context context) throws
IOException, InterruptedException {
            String line = value.toString();
            StringTokenizer tokenizer = new StringTokenizer(line);
            while (tokenizer.hasMoreTokens()) {
                word.set(tokenizer.nextToken());
                context.write(word, one);
            }
        }
     }

     public static class Reduce extends Reducer<Text, IntWritable, Text,
IntWritable> {

        public void reduce(Text key, Iterable<IntWritable> values, Context context)
          throws IOException, InterruptedException {
            int sum = 0;
            for (IntWritable val : values) {
                sum += val.get();
            }
            context.write(key, new IntWritable(sum));
        }
     }

     public static void main(String[] args) throws Exception {
        Configuration conf = new Configuration();

            Job job = new Job(conf, "wordcount");
```

```
        job.setOutputKeyClass(Text.class);
        job.setOutputValueClass(IntWritable.class);

        job.setMapperClass(Map.class);
        job.setReducerClass(Reduce.class);

        job.setInputFormatClass(TextInputFormat.class);
        job.setOutputFormatClass(TextOutputFormat.class);

        FileInputFormat.addInputPath(job, new Path(args[0]));
        FileOutputFormat.setOutputPath(job, new Path(args[1]));

        job.waitForCompletion(true);
    }

}
```

Every MapReduce job consists of three portions

- The driver code
- Code that runs on the client to configure and submit the job
- The Mapper
- The Reducer

Before we look at the code, we need to cover some basic Hadoop API concepts

Mapper reading data from HDFS

The data passed to the Mapper is specified by an InputFormat. The InputFormat is specified in the driver code. It defines the location of the input data like a file or directory on HDFS. It also determines how to split the input data into input splits.

Each Mapper deals with a single input split. InputFormat is a factory for RecordReader objects to extract (key, value) records from the input source.

FileInputFormat is the base class used for all file-based InputFormats. TextInputFormat is the default FileInputFormat. It treats each \n-terminated line of a file as a value. The Key is the byte offset within the file of that line. KeyValueTextInputFormat maps \n-terminated lines as 'key SEP value'. By default, separator is a tab. SequenceFileInputFormat is a binary file of (key, value) pairs with some additional metadata. SequenceFileAsTextInputFormat is similar, but maps (key.toString( ), value.toString( )).

Keys and values in Hadoop are objects. Values are objects which implement the writable interface. Keys are objects which implement writableComparable.

Writable

Hadoop defines its own 'box classes' for strings, integers and so on:

- IntWritable for ints
- LongWritable for longs

- FloatWritable for floats
- DoubleWritable for doubles
- Text for strings
- Etc.

The writable interface makes serialization quick and easy for Hadoop. Any value's type must implement the writable interface.

WritableComparable

A WritableComparable is a Writable which is also Comparable. Two writableComparables can be compared against each other to determine their 'order'. Keys must be WritableComparables because they are passed to the Reducer in sorted order.

Note that despite their names, all Hadoop box classes implement both Writable and WritableComparable, for example, intwritable is actually a WritableComparable

Driver

The driver code runs on the client machine. It configures the job, then submits it to the cluster.

Streaming API

Many organizations have developers skilled in languages other than Java, such as

- C#
- Ruby
- Python
- Perl

The Streaming API allows developers to use any language they wish to write Mappers and Reducers as long as the language can read from standard input and write to standard output.

The advantages of the Streaming API are that there is no need for non-Java coders to learn Java. So it results in faster development time and the ability to use existing code libraries.

How Streaming Works

To implement streaming, write separate Mapper and Reducer programs in the language of your choice. They will receive input via stdin. They should write their output to stdout.

If TextinputFormat (the default) is used, the streaming Mapper just receives each line from the file on stdin where no key is passed. Streaming Mapper and streaming Reducer's output should be sent to stdout as key (tab) value (newline) and the Separators other than tab can be specified.

In Java, all the values associated with a key are passed to the Reducer as an iterator. Using Hadoop Streaming, the Reducer receives its input as (key, value) pairs, one per line of standard input.

Your code will have to keep track of the key so that it can detect when values from a new key start appearing launching a Streaming Job .To launch a Streaming job, use e.g.,:

```
hadoop jar $HADOOP_HOME/contrib/streaming/hadoop-streaming*.jar \
-input mylnputDirs \ -output myOutputDir \
-mapper myMap.py \
-reducer myReduce.py \ -file myMap.py \ -file myReduce.py
```

Repositories

At Hortonworks, we store all of our artifacts in a public Sonatype Nexus repository. That repository can be easily accessed and searched for commonly used library, source code, and javadoc archives simply by navigating to http://repo.hortonworks.com.

Artifacts

Jar files containing compiled classes, source, and javadocs are all available in our public repository, and finding the right artifact with right version is as easy as searching the repository for classes you need to resolve.

For example, If creating a solution that requires the use of a class such as org.apache.hadoop.fs.FileSystem, you can simply search our public repository for the artifact that contains that class using the search capabilities available through http://repo.hortonworks.com. Searching for that class will locate the hadoop-common artifact that is part of the org.apache.hadoop group. There will be multiple artifacts each with a different version.

Artifacts in our repository use a 7 digit version scheme. So if we're looking at the 2.7.1.2.3.2.0-2650 version of this artifact:

- The first three digits (2.7.1) signify the Apache Hadoop base version
- The next four digits (2.3.2.0) signify our Hortonworks Data Platform release
- The final numbers after the hyphen (2650) signifies the build number

As you're looking for the right artifact, it's important to use the artifact version that corresponds to the HDP version you plan to deploy to. You can determine this by using hdp-select versions from the command line, or using Ambari by navigating to Admin > Stack and Versions. If neither of these are available in your version of HDP or Ambari, you can use yum, zypper, or dpkg to query the RPM or Debian packages installed for HDP and note their versions.

Once the right artifact has been found with the version that corresponds to your target HDP environment, it's time to configure your build tool to both resolve our repository and include the artifact as a dependency. The following section outlines how to do both with commonly used with build tools such as Maven, SBT, and Gradle.

FOR DS730: Required reading ends here

Maven Setup

Apache Maven, is an incredibly flexible build tool used by many Hadoop ecosystem projects. In this section we will outline what updates to your project's pom.xml file are required to start resolving HDP artifacts.

Repository Configuration

The pom.xml file enables flexible definition of project dependencies and build procedures. To add the Hortonworks repository to your project, allowing HDP artifacts to be resolved, edit the section and add a entry as

illustrated below:

```
<repositories>

 <repository>

   <id>HDP</id>

   <name>HDP Releases</name>

   <url>http://repo.hortonworks.com/content/repositories/releases/</url>

 </repository>

</repositories>
```

Artifact Configuration

Dependencies are added to Maven using the tag within the section of the pom.xml. To add a dependency such as hadoop-common, add this fragment:

```
<dependency>

 <groupId>org.apache.hadoop</groupId>

 <artifactId>hadoop-common</artifactId>

 <version>2.7.1.2.3.2.0-2650</version>

</dependency>
```

Once both the repository has been added to the section, and the artifacts have been added to the , a simple mvn compile can be issued from the base directory of your project to ensure that proper syntax has been used and the appropriate dependencies are downloaded.

Source & Javadoc

When using Maven with an IDE, it is often helpful to have the accompanying JavaDoc and source code. To obtain both from our repository for the artifacts that you have defined in your pom.xml, run the following commands from the base directory of your project:

```
mvn dependency:sources

mvn dependency:resolve -Dclassifier=javadoc
```

SBT Setup

The Scala Build Tool is commonly used with Scala based projects, and provide simple configuration, and many flexible options for dependency and build management.

Repository Configuration

In order for SBT projects to resolve Hortonworks Data Platform dependencies, an additional resolvers entry must be added to your build.sbt file, or equivalent, as illustrated below:

resolvers += “Hortonworks Releases” at “http://repo.hortonworks.com/content/repositories/releases/”

Artifact Configuration

Dependencies can be added to SBT’s libraryDependencies as illustrated below:

libraryDependencies += “org.apache.hadoop” % “hadoop-common” % “2.7.1.2.3.2.0-2650”

To explicitly ask SBT to also download source code and JavaDocs an alternate notation can be used:

libraryDependencies += “org.apache.hadoop” % “hadoop-common” % “2.7.1.2.3.2.0-2650” withSources() withJavadoc()

Once both the repository has been added to resolvers, and the artifacts have been added to dependencies, a simple sbt compile can be issued from the base directory of your project to ensure that proper syntax has been used and the appropriate dependencies are downloaded.

Gradle Setup

The Gradle build management tool is used frequently in Open Source java projects, and provides a simple Groovy-based DSL for project dependency and build definition.

Plugin Configuration

Gradle uses plugins to add functionality to add new task, domain objects and conventions to your gradle build. Add the following plugins to your build.gradle file, or equivalent, as illustrated below:

```
apply plugin: ‘java’

apply plugin: ‘maven’

apply plugin: ‘idea’  // Pick IDE appropriate for you

apply plugin: ‘eclipse’ // Pick IDE appropriate for you
```

Repository Configuration

In order for Gradle projects to resolve Hortonworks Data Platform dependencies, an additional entry must be added to your build.gradle file, or equivalent, as illustrated below:

```
repositories {

  maven { url “http://repo.hortonworks.com/content/repositories/releases/” }

}
```

Artifact Configuration

Dependencies can be added to Gradle’s dependencies section as illustrated below:

```
dependencies {

   compile group: “org.apache.hadoop”, name: “hadoop-common”, version: “2.7.1.2.3.2.0-2650”

}

idea {  // Pick IDE appropriate for you

module {

  downloadJavadoc = true

  downloadSources = true

}

}

eclipse {  // Pick IDE appropriate for you

classpath {

  downloadSources = true

  downloadJavadoc = true

}

}
```

Once both the repositories and the dependencies have been added to build file, a simple gradle clean build can be issued from the base directory of your project to ensure that proper syntax has been used and the appropriate dependencies are downloaded.

Hive and Pig: Motivation

MapReduce code is typically written in Java. Although it can be written in other languages using Hadoop

Streaming Requires a programmer who understands how to think in terms of MapReduce, who understands the problem they’re trying to solve and who has enough time to write and test the code.

Many organizations have only a few developers who can write good MapReduce code

Meanwhile, many other people want to analyze data

- Data analysts
- Business analysts
- Data scientists
- Statisticians

So we needed a higher-level abstraction on top of MapReduce providing the ability to query the data without needing to know MapReduce intimately. Hive and Pig address these needs.

See the following tutorial for more on Hive and Pig:

- Process Data with Apache Hive
- Process Data with Apache Pig
- Get Started with Cascading on Hortonworks Data Platform
- Interactive Query for Hadoop with Apache Hive on Apache Tez
- Exploring Data with Apache Pig from the Grunt shell

Comments

Chandra Sekhar Gora | October 10, 2014 at 9:03 am | **Reply**

"Process Data with Apache Hive" hyper link is broken @ http://hortonworks.com/hadoop-tutorial/introducing-apache-hadoop-developers/

basically it redirecting to same page.

Thanks,
Chandra Sekhar G,
Sr SDET, apigee,
Bangalore, India.

Saptak Sen | October 10, 2014 at 9:29 am | **Reply**

Thank for for catching it. Now it is fixed.

sanjay malage | December 13, 2014 at 11:08 pm | **Reply**

that was really helpfull to start with hadoop thanks anyways……

Akansha jain | January 12, 2015 at 12:28 pm | **Reply**

hello I m a learner of this technology , could you please tell me which softwares i need for installing the hadoop on windows 8.Please with proper step by step procedure.
Thanks in advance

Sushant | February 11, 2015 at 1:08 pm | **Reply**

Download Hortronworks sandbox and install using Oracle virtual machine and browse thru the tutorial section and you'd be pretty comfortable with the basics.

http://hortonworks.com/sandbox#install

Also register to have delivered training email series in your inbox from Hortonworks to keep consistency of the topics your want to learn.

There also are a lot of youtube videos to work with a sandbox if you get stuck somewhere.

Good luck!

Eduardo | February 12, 2015 at 10:34 pm | **Reply**

Hello, just starting with HDP and Hadoop.
As a developer, I started with this tutorial, but when I want to execute "hadoop jar hadoop-*-examples.jar wordcount [-m] [-r] ", it doesn't explain how and where to do so.
I've been searching for a while in the ssh console with no luck.

Could you tell where is that covered in the tutorials please?

Thank you

Jules S. Damji | April 1, 2015 at 4:40 pm | **Reply**

You can run this command on the Single node cluster, the Hortonworks Sandbox.

paresh | May 2, 2015 at 2:56 pm | **Reply**

Same issue as Eduardo. Is this tutorial supposed to be executable/testable in the Sandbox?

I dont see the jar files or input files anywhere, after SSH as root into the Sandbox 2.2.4 (Virtualbox/Mac).

Thanks

Ludovic | May 9, 2015 at 3:18 am | **Reply**

the jar is in /usr/hdp/2.2.4.2-2/hadoop-mapreduce/hadoop-mapreduce-examples.jar

If you want instead to compile the java class to run your own java class, you have to add the /usr/hdp/2.2.4.2-2/hadoop-mapreduce/* and /usr/hdp/2.2.4.2-2/hadoop/* directories to the classpath

Amir | May 20, 2015 at 12:38 pm | **Reply**

Hi Eduardo, did you figure out how to exactly run the example? where are these directories?

Should WE build the code and move the jar to the Sandbox? how? ftp? etc…Thanks anyone else who can provide detailed guidance.

hadoop jar hadoop-*-examples.jar wordcount [-m] [-r]

Aniket Savanand | February 16, 2015 at 2:08 pm | **Reply**

Thank you.

bhavana | March 3, 2015 at 9:20 am | **Reply**

hello i m a learner of this technology,could you please tell me the steps for installing the hadoop in single node and multinode?

Sarra | March 5, 2015 at 1:31 am | **Reply**

Hi, can I continue developping with PHP and using MySQL if I install HDP??

Ramesh | April 20, 2015 at 1:43 pm | **Reply**

Hi..I Have downloaded HDP for Windows. Shall i install the Hortonworks sandbox in windows 7. Please let me know the quick guide to go through on it. Thanks.

Pramod Mudgal | April 23, 2015 at 12:49 am | **Reply**

Where can i fiind the source code of hadoop-*-examples.jar used in this tutorial.

Ludovic | May 9, 2015 at 3:12 am | **Reply**

To correctly run this example:

Install the Sandbox and make sure HDFS and MapReduce are activated
You can either
Run the hadoop original example from */usr/hdp/2.2.4.2-2/hadoop-mapreduce/hadoop-mapreduce-examples.jar* (the version may vary with your HDP virtual machine version)
Create your own WordCount.java

In the case you want to create your own WordCount class

Copy/paste the source code from this tutorial
Locate the line *Job job = new Job(conf, "wordcount");* and add
job.setJarByClass(WordCount.class); just after to specify Hadoop which code execute in the distributed nodes (even if you just have one)
Compile it *javac -cp /usr/hdp/2.2.4.2-2/hadoop/*:/usr/hdp/2.2.4.2-2/hadoop-mapreduce/* WordCount.java -d build -Xlint*
Build the jar *jar -cvf wordcount.jar -C build/ .*

Run the jar with hadoop*hadoop jar wordcount.jar org.myorg.WordCount /user/test/wordcount/input /user/test/wordcount/output*

Before doing this, you must insert some text files into HDFS. To do this,
connect to the HDP (ssh or directly from vmware/virtualbox)
create a test user (as root): #**useradd test**
create the test user directory in hadoop as hdfs user : **#sudo su hdfs$hadoop fs -mkdir /user/test$hadoop fs -chown test /user/test & exit**
create the input directory in hadoop directory as test user : **#sudo su test$hadoop fs -mkdir /user/test/wordcount /user/test/wordcount/input**
Put some local text files into hadoop (as test user) : **$hadoop dfs -copyFromLocal /tmp/sometextfiles /user/test/wordcount/input** (to create local text files, something like $echo "I'm a text">/tmp/file create a file *file* containing "I'm a text". I recommend you load more than one file into hadoop to make this example a bit more exciting)
Now everything should be OK to run the example : **hadoop jar wordcount.jar org.myorg.WordCount /user/test/wordcount/input /user/test/wordcount/output**
To see the result: **$hadoop fs -cat /user/test/wordcount/output/***
And to run the example again, you have first to remove the ouput directory : **hadoop fs -rm -f -R /user/test/wordcount/output**

Peter Trei | July 21, 2015 at 2:46 pm | **Reply**

Ludovic: Thanks a million – your comment actually made this tutorial workable.
pt

tubelugs | August 12, 2015 at 11:07 am | **Reply**

While I appreciate the time to write all this up, most of the commands did not work for me. useradd test simply produced a new bash prompt–no feedback on whether the command worked or not. When I try to -l test I get a reply that -l is not a recognized command. (?What?) The command to create the test user directory logs me out and puts me at the bash login prompt.

Having attempted to work through a number of HortonWorks tutorials without success – there's always something assumed that isn't explained, or something missing – I am quickly coming to the conclusion that HortonWorks HDP is not ready for prime time.

Paul Praet | September 2, 2015 at 12:34 am | **Reply**

Thanks a lot, Ludovic. Why can't hortonworks give a decent description how to run the samples ??

anthony | August 9, 2015 at 10:25 am | **Reply**

Thx

anthony | August 9, 2015 at 10:30 am | **Reply**

Yee

amit | August 13, 2015 at 12:17 am | **Reply**

attempted to work through a number of HortonWorks tutorials without success – help is missing every tutorials — 🙁

Joe | August 15, 2015 at 8:38 am | **Reply**

question on the sandbox, I had downloaded, and imported the image. Do I need to install the Hadoop on the sandbox? I couldn't find the /user and all the examples that suppose to come with hadoop in the sandbox environment. Can someone help me? Thanks!

anoop | September 2, 2015 at 7:55 pm | **Reply**

you need to just import the image on your virtual box or vm player based on the sandbox you have downloaded. It has everything in it to start with…

sathish | October 11, 2015 at 1:52 am | **Reply**

how to debug mapreduce program in eclipse

sathish | October 27, 2015 at 10:03 pm | **Reply**

I am using Hortonworks sandbox for hadoop …How do i debug java code in eclipse?

I am using Horton works sandbox for hadoop, But i am using eclipse on windows for the java code . and hortonworks sanbox installed in oracle virtual box. I created executable jar for that java code and then run in hortons. How do I debug the java code in eclipse?

akshay naidu | November 25, 2015 at 12:00 am | **Reply**

can i run these tutorials in HDP 1.3

alboko | December 8, 2015 at 8:20 am | **Reply**

SBT doesn't seem to handle dependencies correctly for HBT artifacts. If you want to build the examples with SBT you should add the following to your build.sbt:

```
resolvers += "Hortonworks Releases" at "http://repo.hortonworks.com/content
/repositories/releases/"
resolvers += "jetty-hadoop" at "http://repo.hortonworks.com/content
```

```
/repositories/jetty-hadoop/"

 libraryDependencies ++= Seq(
"org.apache.hadoop" % "hadoop-common" % "2.7.1.2.3.2.0-2950",
"org.apache.hadoop" % "hadoop-mapreduce" % "2.7.1.2.3.2.0-2950",
"org.apache.hadoop" % "hadoop-mapreduce-client-core" % "2.7.1.2.3.2.0-2950"
)
```

replace artifact version “2.7.1.2.3.2.0-2950” by the version of your sandbox

Sushant | December 9, 2015 at 12:37 am | **Reply**

Hi,
I need to implement Expectation Maximization and Minimal spanning tree clustering algorithm in hadoop please let me know where to start, I have installed sandbox but I am clueless from here.

CONTACT

Contact

+1 408 675-0983

+1 855 8-HORTON

INTL: +44 (0) 20 3826 1405