

- » Design Concepts
- » Hadoop Components
- » HDFS
- » YARN
- » YARN Applications
- » MapReduce
- » And More...

Getting Started with Apache Hadoop

By Adam Kawa and Piotr Krewski

INTRODUCTION

This Refcard presents Apache Hadoop, a software framework that enables distributed storage and processing of large datasets using simple high-level programming models. We cover the most important concepts of Hadoop, describe its architecture, guide how to start using it as well as write and execute various applications on Hadoop.

In the nutshell, Hadoop is an open-source project of the Apache Software Foundation that can be installed on a set of standard machines, so that these machines can communicate and work together to store and process large datasets. Hadoop has become very successful in recent years thanks to its ability to effectively crunch big data. It allows companies to store all of their data in one system and perform analysis on this data that would be otherwise impossible or very expensive to do with traditional solutions.

Many companion tools built around Hadoop offer a wide variety of processing techniques. Integration with ancillary systems and utilities is excellent, making real-world work with Hadoop easier and more productive. These tools together form the Hadoop Ecosystem.

HOT TIP

Visit <http://hadoop.apache.org> to get more information about the project and access detailed documentation.

Note: By a standard machine, we mean typical servers that are available from many vendors and have components that are expected to fail and be replaced on a regular base. Because Hadoop scales nicely and provides many fault-tolerance mechanisms, you do not need to break the bank to purchase expensive top-end servers to minimize the risk of hardware failure and increase storage capacity and processing power.

DESIGN CONCEPTS

To solve the challenge of processing and storing large datasets, Hadoop was built according to the following core characteristics:

- Distribution - instead of building one big supercomputer, storage and processing are spread across a cluster of smaller machines that communicate and work together.
- Horizontal scalability - it is easy to extend a Hadoop cluster by just adding new machines. Every new machine increases total storage and processing power of the Hadoop cluster.
- Fault-tolerance - Hadoop continues to operate even when a few hardware or software components fail to work properly.
- Cost-optimization - Hadoop runs on standard hardware; it does not require expensive servers.
- Programming abstraction - Hadoop takes care of all messy details related to distributed computing. Thanks to a high-level API, users can focus on implementing business logic that solves their real-world problems.
- Data locality - don't move large datasets to where application is running, but run the application where the data already is.

HADOOP COMPONENTS

Hadoop is divided into two core components

- HDFS - a distributed file system
- YARN - a cluster resource management technology

HOT TIP

Many execution frameworks run on top of YARN, each tuned for a specific use-case. The most important are discussed under 'YARN Applications' below.

Let's take a closer look on their architecture and describe how they cooperate.

Note: YARN is the new framework that replaces the former implementation of the processing layer in Hadoop. You can find how YARN addresses shortcomings of previous version on the Yahoo blog: <https://developer.yahoo.com/blogs/hadoop/next-generation-apache-hadoop-mapreduce-3061.html>.

HDFS

HDFS is a Hadoop distributed file system. It can be installed on commodity servers and run on as many servers as you need - HDFS easily scales to thousands of nodes and petabytes of data.

The larger HDFS setup is, the bigger probability that some disks, servers or network switches will fail. HDFS survives these types of failures by replicating data on multiple servers. HDFS automatically detects that a given component has failed and takes necessary recovery actions that happen transparently to the user.

HDFS is designed for storing large files of the magnitude of hundreds of megabytes or gigabytes and provides high-throughput streaming data access to them. Last but not least, HDFS supports the write-once-read-many model. For this use case HDFS works like a charm. If you need, however, to store a large number of small files with a random read-write access, then other systems like RDBMS and Apache HBase can do a better job.

Note: HDFS does not allow you to modify a file's content. There is only support for appending data at the end of the file. However, Hadoop was designed with HDFS to be one of many pluggable storage options - for example, with MapR-Fs, a proprietary filesystem, files are fully read-write. Other HDFS alternatives include Amazon S3 and IBM GPFS.

ARCHITECTURE OF HDFS

HDFS consists of following daemons that are installed and run on selected cluster nodes:

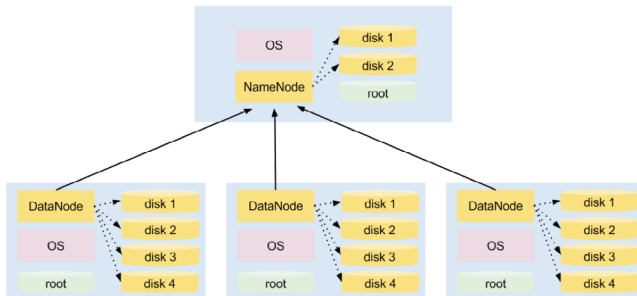
MAPR Sandbox

Take the Fastest On-Ramp to
Hadoop with the MapR Sandbox

Try the sandbox FREE today at www.mapr.com/sandbox.

- NameNode - the master process responsible for managing the file system namespace (filenames, permissions and ownership, last modification date etc.) and controlling access to data stored in HDFS. It is the one place where there is a full overview of the distributed file system. If the NameNode is down, you can not access your data. If your namespace is permanently lost, you've essentially lost all of your data!
- DataNodes - slave processes that take care of storing and serving data. A DataNode is installed on each worker node in the cluster.

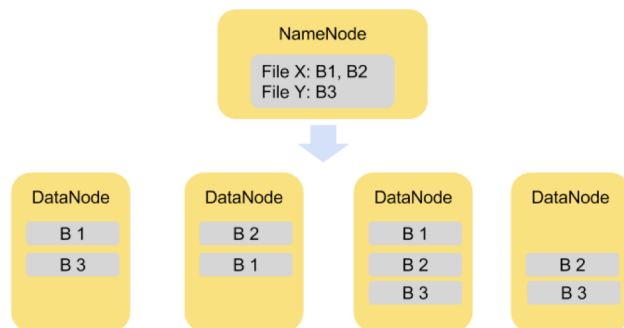
Figure 1 illustrates installation of HDFS on a 4-node cluster. One of the node hosts the NameNode daemon while the other three run DataNode daemons.



Note: NameNode and DataNode are Java processes that run on top of Linux distribution such as RedHat, Centos, Ubuntu and more. They use local disks for storing HDFS data.

HDFS splits each file into a sequence of smaller, but still large, blocks (default block size equals to 128MB – bigger blocks mean fewer disk seek operations, which results in large throughput). Each block is stored redundantly on multiple DataNodes for fault-tolerance. The block itself does not know which file it belongs to – this information is only maintained by the NameNode that has a global picture of all directories, files and blocks in HDFS.

Figure 2 illustrates the concept of splitting files into blocks. File X is split into blocks B1 and B2 and File Y comprises of only one block B3. All blocks are replicated 2 times within the cluster. As mentioned, information about which blocks compose a file is kept by the NameNode while raw data is stored by DataNodes.



INTERACTING WITH HDFS

HDFS provides a simple POSIX-like interface to work with data. You perform file system operations using `hdfs dfs` command.

HOT TIP

To start playing with Hadoop you don't have to go through the process of setting up a whole cluster. Hadoop can run in so called pseudo-distributed mode on a single machine. You can download the sandbox Virtual Machine with all HDFS components already installed and start using Hadoop in no time! Just follow one of these links:

<http://www.mapr.com/products/mapr-sandbox-hadoop>

<http://hortonworks.com/products/ Hortonworks-sandbox/#install> http://www.cloudera.com/content/support/en/downloads/quickstart_vms/cdh-5-1-x1.html

http://www.cloudera.com/content/support/en/downloads/quickstart_vms/cdh-5-1-x1.html

The following steps illustrate typical operations that a HDFS user can perform:

```
List the content of home directory
$ hdfs dfs -ls /user/adam

Upload a file from local file system to HDFS
$ hdfs dfs -put songs.txt /user/adam

Read the content of the file from HDFS
$ hdfs dfs -cat /user/adam/songs.txt

Change the permission of a file
$ hdfs dfs -chmod 700 /user/adam/songs.txt

Set the replication factor of a file to 4
$ hdfs dfs -setrep -w 4 /user/adam/songs.txt

Check the size of the file
$ hdfs dfs -du -h /user/adam/songs.txt

Create subdirectory in your home directory
$ hdfs dfs -mkdir songs

Move the file to the newly created subdirectory
$ hdfs dfs -mv songs.txt songs/

Remove directory from HDFS
$ hdfs dfs -rm -r songs
```

HOT TIP

You can type `hdfs dfs` without any parameters to get a full list of available commands

YARN

YARN is a framework that manages resources on the cluster and enables running various distributed applications that process data stored (usually) on HDFS.

YARN, similarly to HDFS, follows the master-slave design with single ResourceManager daemon and multiple NodeManagers daemons. These types of daemons have different responsibilities.

ResourceManager

- keeps track of live NodeManagers and the amount of available compute resources that they currently have
- allocates available resources to applications submitted by clients
- monitors whether applications complete successfully

NodeManagers

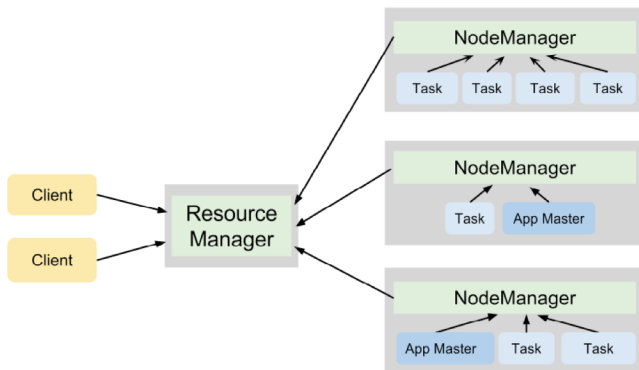
- offer computational resources in form of containers
- run various applications' tasks inside the containers

YARN assigns cluster resources to various applications in the form of resource containers which represent a combination of resource elements such as memory and CPU.

Each application that executes on YARN cluster has its own ApplicationMaster process. This process starts when the application is scheduled on the cluster and coordinates the execution of all tasks within this application.

Each task runs within a container managed by the selected NodeManager. The ApplicationMaster negotiates resources (in form of containers) with the ResourceManager. On a successful negotiation, the ResourceManager delivers a container specification to the ApplicationMaster. This specification is then handed over to a NodeManager which launches the container and executes a task within it.

Figure 3 illustrates cooperation of YARN daemons on 4-node cluster running two applications that spawned 7 tasks in total.



HADOOP 2.0 = HDFS + YARN

HDFS and YARN daemons running on the same cluster give us a powerful platform for storing and processing large datasets.

Interestingly, DataNode and NodeManager processes are collocated on the same nodes to enable one of the biggest advantages of Hadoop called data locality. Data locality allows us to perform computations on the machines that actually store the data, thus minimizing the necessity of sending large chunks of data over the network. This technique known as “sending computation to the data” causes significant performance improvements while processing large data.

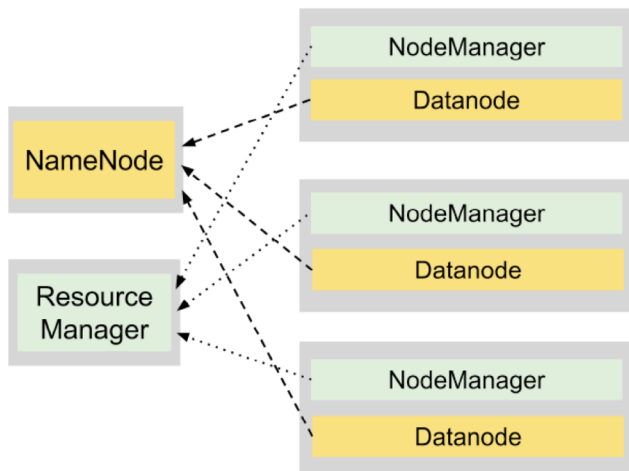


Figure 4: collocating HDFS and YARN daemons on a Hadoop cluster.

YARN APPLICATIONS

YARN is merely a resource manager that knows how to allocate distributed compute resources to various applications running on a Hadoop cluster. In other words, YARN itself does not provide any processing logic that can analyze data in HDFS. Hence various processing frameworks must be integrated with YARN (by providing a specific implementation of the ApplicationMaster) to run on a Hadoop cluster and process data in HDFS.

The table below provides a list and short descriptions of the most popular distributed computation frameworks that can run on a Hadoop cluster powered by YARN.

MapReduce	The most popular processing framework for Hadoop that expresses computation as a series of map and reduce tasks. MapReduce is explained in the next section.
-----------	--

Apache Spark	A fast and general engine for large-scale data processing that optimizes the computation by aggressively caching data in memory.
Apache Tez	Generalizes the MapReduce paradigm to a more powerful and faster framework that executes computation as complex directed acyclic graphs of general data processing tasks.
Apache Giraph	An iterative graph processing framework for big data.
Apache Storm	A realtime stream processing engine.
Cloudera Impala	Fast SQL on Hadoop.

MAPREDUCE

MapReduce is a programming model that allows for implementing parallel distributed algorithms. To define computations in this paradigm you provide the logic for two functions: map() and reduce() that operate on <key, value> pairs.

Map function takes a <key,value> pair and produces zero or more intermediate <key, value> pairs:

$\text{Map}(k1, v1) \rightarrow \text{List}(k2, v2)$

Reduce function takes a key and list of values associated with this key and produces zero or more final <key, value> pairs:

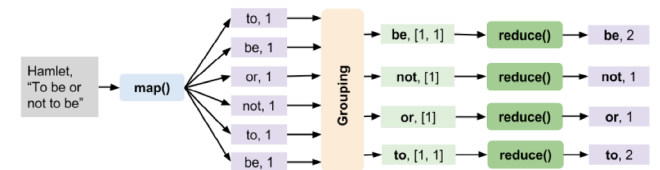
$\text{Reduce}(k2, \text{List}(v2)) \rightarrow \text{List}(k3, v3)$

Between Map and Reduce functions all intermediate <key, value> pairs produced by Map functions are shuffled and sorted by key, so that all the values associated with the same key are grouped together and passed to the same Reduce function.



The general purpose of Map function is to transform or filter the input data. On the other hand Reduce function typically aggregates or summarizes the data produced by Map functions.

Figure 6 shows an example of using MapReduce to count occurrences of distinct words in a sentence. Map function splits the sentence and produces intermediate <key, value> pairs where a key is the word and a value equals to 1. Then reduce function sums all the 1s associated with a given word returning the total number of occurrences of that word.



MAPREDUCE ON YARN

MapReduce on YARN is a framework that enables running MapReduce jobs on the Hadoop cluster powered by YARN. It provides a high-level API for implementing custom Map and Reduce functions in various languages as well as the code-infrastructure needed to submit, run and monitor MapReduce jobs.

Note: MapReduce was historically the only programming model that you

could use with Hadoop. It is no longer the case after the introduction of YARN. MapReduce is still the most popular application running on YARN clusters, though.

The execution of each MapReduce job is managed and coordinated by an instance of a special process called MapReduce Application Master (MR AM). MR AM spawns Map tasks that runs `map()` functions and Reduce tasks that run `reduce()` functions. Each Map task processes a separate subset of the input dataset (one block in HDFS by default). Each reduce task processes a separate subset of the intermediate data produced by the Map tasks. What's more, Map and Reduce tasks run in isolation from one another, which allows for parallel and fault-tolerant computations.

To optimize the computation, MR AM tries to schedule data-local Map tasks. Such tasks execute in the containers running on the NodeManagers that are collocated with DataNodes that already store the data we want to process. Because by default each block in HDFS is redundantly stored on three DataNodes there are three NodeManagers that can be asked to run a given Map task locally.

SUBMITTING A MAPREDUCE JOB

Let's see MapReduce in action and run a MapReduce job on a Hadoop cluster.

To get started quickly we use a jar file with MapReduce examples that is supplied with Hadoop packages. On Linux systems it can be found under

`/usr/lib/hadoop-mapreduce/hadoop-mapreduce-examples.jar`

We run the Word Count job explained in the previous section.

1. Create a file named `hamlet.txt` that has following content:

```
To be or not to be
```

2. Upload input data on HDFS

```
# hdfs dfs -mkdir input
# hdfs dfs -put hamlet.txt input/
```

3. Submit the WordCount MapReduce job to the cluster:

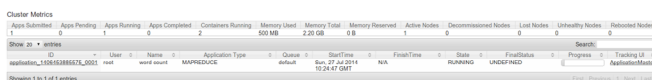
```
# hadoop jar /usr/lib/hadoop-mapreduce/hadoop-mapreduce-examples.jar wordcount input hamlet-output
```

After a successful submission, track the progress of this job on the ResourceManager web UI.

HOT TIP

If you use a sandbox, the ResourceManager UI is available at <http://localhost:8088>

Figure 7: ResourceManager UI with running Job



Cluster Metrics									
Apps Submitted	Apps Pending	Apps Running	Apps Completed	Containers Running	Memory Used	Memory Total	Memory Reserved	Active Nodes	Decommissioned Nodes
1	0	2	0	2	500 MB	2.0 GB	0 B	1	0

Job ID	User	Name	Application Type	Owner	Start Time	Final Time	State	Final Status	Progress	Tracking UI
application_140593885621_0001	root	wordcount	MAPREDUCE	adam	Sun, 27 Jul 2014 10:24:42 GMT	N/A	RUNNING	UNDEFINED	0%	ApplicationMaster

4. Check the output of this job in HDFS:

```
# hadoop fs -cat hamlet-output/*
```

Apart from the Word Count job, the jar file contains several other MapReduce examples. You can list them by typing the following command:

```
# hadoop jar /usr/lib/hadoop-mapreduce/hadoop-mapreduce-examples.jar
```

The table below provides a list and short descriptions of a couple of interesting MapReduce examples:

grep	Counts the matches of a given regular expression in the input dataset.
pi	Estimates Pi using a quasi-Monte Carlo method.
terasort	Sorts the input dataset. Often used in conjunction with teragen and teravalidate. Find more details at https://hadoop.apache.org/docs/current/api/org/apache/hadoop/examples/terasort/package-summary.html
wordmean	Counts the average length of the words in the input dataset.

PROCESSING FRAMEWORKS

Developing applications in native MapReduce can be a time-consuming and daunting work reserved only for programmers.

Fortunately, there are a number of frameworks that make the process of implementing distributed computation on Hadoop cluster easy and quicker, even for non-developers. The most popular ones are Hive and Pig.

HIVE

Hive provides a SQL-like language, called HiveQL, for easier analysis of data in Hadoop cluster. When using Hive our datasets in HDFS are represented as tables that have rows and columns. Therefore, Hive is easy to learn and appealing to use for those who already know SQL and have experience in working with relational databases.

Having this said, Hive can be considered as a data warehouse infrastructure built on top of Hadoop.

A Hive query is translated into a series of MapReduce jobs (or a Tez directed acyclic graph) that are subsequently executed on a Hadoop cluster.

Hive example

Let's process a dataset about songs listened to by users in a given time. The input data consists of a tab-separated file `songs.txt`:

```
"Creep" Radiohead piotr 2014-07-20
"Desert Rose" Sting adam 2014-07-14
"Desert Rose" Sting piotr 2014-06-10
"Karma Police" Radiohead adam 2014-07-23
"Everybody" Madonna piotr 2014-07-01
"Stupid Car" Radiohead adam 2014-07-18
"All This Time" Sting adam 2014-07-13
```

We use Hive to find the two most popular artists in July 2014:

Note: We assume that commands below are executed as user "training".

1. Put `songs.txt` file on HDFS:

```
# hdfs dfs -mkdir songs
# hdfs dfs -put songs.txt songs/
```

2. Enter hive:

```
# hive
hive>
```

3. Create an external table in Hive that gives a schema to our data on HDFS:

```
hive> CREATE TABLE songs(
  title STRING,
  artist STRING,
  user STRING,
  date DATE
)
ROW FORMAT DELIMITED
FIELDS TERMINATED BY '\t'
LOCATION '/user/training/songs';
```

4. Check if the table was created successfully:

```
hive> SHOW tables;
```

5. You can see also the table's properties and columns:

Apart from information about column names and types, you can see other interesting properties:

```
# Detailed Table Information

Database:      default

Owner:         root

CreateTime:    Tue Jul 29 14:08:49 PDT 2014
LastAccessTime: UNKNOWN

Protect Mode:  None

Retention:     0

Location:      hdfs://localhost:8020/user/root/songs
Table Type:    EXTERNAL_TABLE
```

Among other things, notice that the table was created in the default database, where all new tables are placed if you do not specify a custom database. There is also an entry with an HDFS path, so you can easily locate the dataset that backs your table.

6. Run a query that finds the two most popular artists in July 2014:

```
SELECT artist, COUNT(*) AS total
FROM songs
WHERE year(date) = 2014 AND month(date) = 7
GROUP BY artist
ORDER BY total DESC
LIMIT 2;
```

This query is translated to two MapReduce jobs. Verify it by reading the standard output log messages generated by a Hive client or by tracking jobs executed on Hadoop cluster using ResourceManager web UI.

Note: at the time of this writing, MapReduce was the default execution engine for Hive. It may change in the future. See next section for instructions how to set other execution engine for Hive.

TEZ

Hive is not constrained to translate queries into MapReduce jobs only. You can also instruct Hive to express its queries using other distributed frameworks such as Apache Tez.

Tez is an efficient framework that executes computation in form of a DAG (directed acyclic graph) of tasks. With Tez, a complex Hive query can be expressed as a single Tez DAG rather than multiple MapReduce jobs. This way we do not introduce the overhead of launching multiple jobs and avoid the cost of storing data between jobs on HDFS what saves I/O.

To benefit from Tez's fast response times, simply overwrite hive.execution.engine property and set it to tez.

Follow these steps to execute the Hive query from the previous section as a Tez application:

1. Enter hive:

```
# hive
hive>
```

2. Set execution engine to tez:

```
hive> SET hive.execution.engine=tez;
```

3. Execute query from the Hive section:

Note: now you can see different logs displayed on the console than when executing the query on MapReduce:

```
Total Jobs = 1
Launching Job 1 out of 1

Status: Running application id: application_123123_0001

Map 1: -- Reducer 2: 0/1 Reducer 3: 0/1
Map 1: 0/1 Reducer 2: 0/1 Reducer 3: 0/1
...
Map 1: 1/1/ Reducer 2: 1/1 Reducer 3: 1/1
Status: Finished successfully
OK
Radiohead 3
Sting 2
```

The query is now executed as only one Tez job instead of two MapReduce jobs as before. Tez isn't tied to a strict MapReduce model - it can execute any sequence of tasks in a single job, for example Reduce tasks after Reduce tasks, what brings significant performance benefits.

HOT TIP

Find out more about Tez on the blog: <http://hortonworks.com/blog/apache-tez-a-new-chapter-in-hadoop-data-processing>.

PIG

Apache Pig is another popular framework for large-scale computations on Hadoop. Similarly to Hive, Pig allows you to implement computations in an easier, faster and less-verbose way than using MapReduce. Pig introduces a simple, yet powerful, scripting-like language called PigLatin. PigLatin supports many common and ready-to-use data operations like filtering, aggregating, sorting and joining. Developers can also implement own functions (UDFs) that extend Pig's core functionality.

Like Hive queries, Pig scripts are translated to MapReduce jobs scheduled to run on Hadoop cluster.

We use Pig to find the most popular artists as we did with Hive in previous example.

1. Save following script in top-artists.pig file

```
a = LOAD 'songs/songs.txt' as (title, artist, user, date);
b = FILTER a BY date MATCHES '2014-07-*';
c = GROUP b BY artist;
d = FOREACH c GENERATE group, COUNT(b) AS total;
e = ORDER d by total DESC;
f = LIMIT e 2;
STORE f INTO 'top-artists-pig';
```

2. Execute pig script on Hadoop cluster:

```
# pig top-artists.pig
```

3. Read the content of the output directory:

```
# hadoop fs -cat top-artists-pig/*
```


HOT TIP

When developing Pig scripts you can iterate in local mode and catch mistakes before submitting jobs to the cluster. To enable local mode add -x local option to pig command.

SUMMARY

Apache Hadoop is one of the most popular tools for big data processing thanks to its great features such as a high-level API, scalability, the ability to run on commodity hardware, fault tolerance and an open source nature. Hadoop has been successfully deployed in production by many companies for several years.

The Hadoop Ecosystem offers a variety of open-source tools for collecting, storing and processing data as well as cluster deployment, monitoring and data security. Thanks to this amazing ecosystem of tools, each company can now easily and relatively cheaply store and process a large amount of data in a distributed and highly scalable way.

HADOOP ECOSYSTEM

This table contains names and short descriptions of the most useful and popular projects from the Hadoop Ecosystem that have not been mentioned yet:

Oozie	Workflow scheduler system to manage Hadoop jobs.
Zookeeper	Framework that enables highly reliable distributed coordination.
Sqoop	Tool for efficient transfer of bulk data between Hadoop and structured datastores such as relational databases.
Flume	Service for aggregating, collecting and moving large amounts of log data.

Hbase

Non-relational, distributed database running on top of HDFS. It enables random realtime read/write access to your Big Data.

ADDITIONAL RESOURCES

- Apache Hadoop: <http://hadoop.apache.org/>
- Apache Hive: <https://hive.apache.org/>
- Apache Pig: <http://pig.apache.org/>
- Apache Giraph: <http://giraph.apache.org/>
- Apache Mahout (machine learning on Hadoop): <https://mahout.apache.org/>
- Apache Tez: <http://tez.apache.org/>
- Apache Spark: <https://spark.apache.org/>
- Apache Storm: <https://storm.incubator.apache.org/>
- Major packaged distributions:
 - o Cloudera: <http://www.cloudera.com/content/cloudera/en/products-and-services/cdh.html>
 - o MapR: <https://www.mapr.com/products/mapr-editions>
 - o Hortonworks: <http://hortonworks.com/hadoop/>

ABOUT THE AUTHOR



Piotr Krewski is passionate about Big Data technologies and has been working with Apache Hadoop in multiple companies. He has extensive practical experience in writing applications running on Hadoop clusters as well as in maintaining, managing and expanding Hadoop clusters. He is a co-founder of GetInData.com where he serves also as Hadoop instructor and consultant. Piotr holds Msc in Computer Science from Warsaw University.



Adam Kawa has extensive practical experience in operating large Hadoop clusters that use several projects from Hadoop ecosystem. Currently, at Spotify he maintains one of the largest and fastest-growing Hadoop clusters in Europe. Adam is a co-founder of [GetInData](http://GetInData.com) where he works as a Hadoop instructor and consultant. He is a frequent speaker at Hadoop conferences. He blogs about Hadoop at HakunaMapData.com and IBM developerWorks.

RECOMMENDED BOOK



Hadoop: The Definitive Guide is the best comprehensive, book-length introduction to building and maintaining distributed, scalable applications with Hadoop. Suitable for both developers working with big data and administrators of Hadoop clusters, this book covers everything from MapReduce and HDFS basics to large-scale data loading, processing, and integration with the rest of the extensive Hadoop ecosystem.

BUY NOW

BROWSE OUR COLLECTION OF **250+ FREE RESOURCES**, INCLUDING:

RESEARCH GUIDES: Unbiased insight from leading tech experts

REFCARDZ: Library of 200+ reference cards covering the latest tech topics

COMMUNITIES: Share links, author articles, and engage with other tech experts

JOIN NOW

DZone, Inc.
 150 Preston Executive Dr.
 Suite 201
 Cary, NC 27513
 888.678.0399
 919.678.0300

Refcardz Feedback Welcome
refcardz@dzone.com
 Sponsorship Opportunities
sales@dzone.com

ISBN-13: 978-1-936502-77-6
 ISBN-10: 1-936502-77-1



Version 1.0 \$7.95



DZone communities deliver over 6 million pages each month to more than 3.3 million software developers, architects and decision makers. DZone offers something for everyone, including news, tutorials, cheat sheets, research guides, feature articles, source code and more.

"DZone is a developer's dream," says PC Magazine.

Copyright © 2014 DZone, Inc. All rights reserved. No part of this publication may be reproduced, stored in a retrieval system, or transmitted, in any form or by means electronic, mechanical, photocopying, or otherwise, without prior written permission of the publisher.