GETTING STARTED WITH

Apache Hadoop

WRITTEN BY PIOTR KREWSKI, FOUNDER AND BIG DATA CONSULTANT AT GETINDATA
AND, **ADAM KAWA**, CEO AND FOUNDER AT GETINDATA

CONTENTS

INTRODUCTION

This Refcard presents Apache Hadoop, the most popular software framework enabling 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 you on how to start using it as well as write and execute various applications on Hadoop.

In a nutshell, Hadoop is an open-source project of the Apache Software Foundation that can be installed on a cluster of servers so that these servers 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.

You can think of Hadoop as a Big Data Operating System that makes it possible to run different types of workloads over all your huge datasets. This ranges from offline batch processing through machine learning to real-time stream processing.

HOT TIP

Visit hadoop.apache.org to get more information about the project and access detailed documentation.

To install Hadoop, you can take the code from **hadoop.apache.org** or (what is more recommended) use one of the Hadoop distributions. The three most widely used ones come from Cloudera (CDH), Hortonworks (HDP), and MapR. Hadoop distribution is a set of tools from the Hadoop ecosystem bundled together and guaranteed by the respective vendor that work and integrate with each other well. Additionally, each vendor offers tools (open-source or proprietary) to provision, manage, and monitor the whole platform.

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 proportionally increases the 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 does not require expensive top-end servers and can work just fine without commercial licenses.
- **Programming abstraction** - Hadoop takes care of all the 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 - Hadoop doesn't move large datasets to where the application is running, but runs 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 at their architecture and describe how they cooperate.

HDFS

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

The larger the HDFS setup is, the bigger the 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 the 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. However, if you need to store a large number of small files with random read-write access, then other systems like RDBMS and Apache HBase can do a better job.

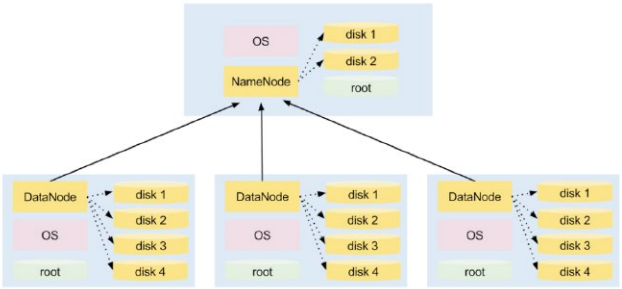
Note: HDFS does not allow you to modify a file's content. There is only support for appending data at the end of the file. However, Hadoop was designed with HDFS to be one of many pluggable storage options — for example, with MapR-Fs, a proprietary filesystem, files are fully read-write. Other HDFS alternatives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

ARCHITECTURE OF HDFS

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

- NameNode - the master process that is responsible for managing the file system namespace (filenames, permissions and ownership, last modification date, etc.) and controlling access to data stored in HDFS. If the NameNode is down, you cannot access your data. Fortunately, you can configure multiple NameNodes that ensure high availability for this critical HDFS process.
- **DataNodes** - slave processes installed on each worker node in the cluster that take care of storing and serving data.

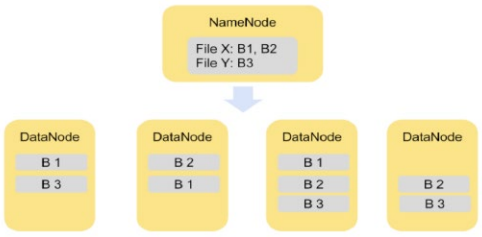
Figure 1 illustrates installation of HDFS on a 4-node cluster. One of the nodes hosts the NameNode daemon while the other three run DataNode daemons.



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

HDFS splits each file into a sequence of smaller, but still large, blocks (the default block size equals 128MB — bigger blocks mean fewer disk seek operations, which results in larger throughput). Each block is stored redundantly on three DataNodes for fault-tolerance (the number of replicas for each file is configurable).

Figure 2 illustrates the concept of splitting files into blocks. File X is split into blocks B1 and B2 and File Y comprises only one block, B3. All blocks are replicated twice within the cluster.



INTERACTING WITH HDFS

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

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 the HDFS components already installed and start using Hadoop in no time! Just follow one of these links:

- mapr.com/products/mapr-sandbox-hadoop
- hortonworks.com/products/hortonworks-sandbox/#install
- cloudera.com/downloads/quickstart_vms/5-12.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 the 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 a file:

```
`$ hdfs dfs -du -h /user/adam/songs.txt
```

Create a subdirectory in your home directory.

```
$ hdfs dfs -mkdir songs
```

Note that relative paths always refer to the home directory of the user executing the command. There is no concept of a "current" directory on HDFS (in other words, there is no equivalent to the "cd" command):

Move the file to the newly created subdirectory:

```
$ hdfs dfs -mv songs.txt songs
```

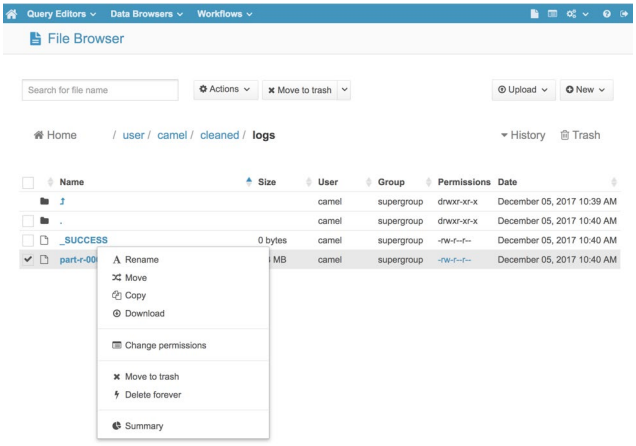
Remove a directory from HDFS:

```
$ hdfs dfs -rm -r songs
```

Note: Removed files and directories are moved to the trash (.Trash in your home directory on HDFS) and stay for one day until they are permanently deleted. You can restore them just by copying or moving them from .Trash to the original location.

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

If you prefer to use a graphical interface to interact with HDFS, you can have a look at free and open-source HUE (Hadoop User Experience). It contains a convenient "File Browser" component that allows you to browse HDFS files and directories and perform basic operations.



You can also use HUE to upload files to HDFS directly from your computer with the "Upload" button.

YARN

YARN (Yet Another Resource Negotiator) is responsible for managing resources on the Hadoop cluster and enables running various distributed applications that process data stored on HDFS.

YARN, similarly to HDFS, follows the master-slave design with the ResourceManager process acting as a master and multiple NodeManagers acting as workers. They have the following responsibilities:

ResourceManager

- Keeps track of live NodeManagers and the amount of available compute resources on each server in the cluster.
- Allocates available resources to applications.
- Monitors execution of all the applications on the Hadoop cluster.

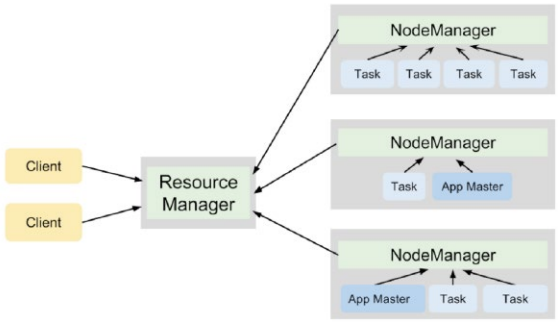
NodeManager

- Manages compute resources (RAM and CPU) on a single node in the Hadoop cluster.
- Runs various applications' tasks and enforces that they are within the limits of assigned compute resources.

YARN assigns cluster resources to various applications in the form of resource containers, which represent a combination of the amount of RAM and number of CPU cores.

Each application that executes on the 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.

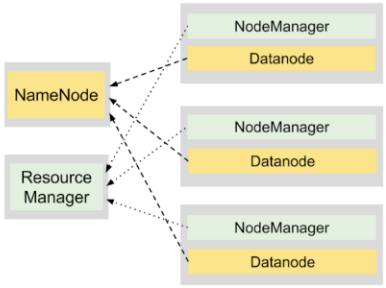
Figure 3 illustrates the cooperation of YARN daemons on a 4-node cluster running two applications that spawned 7 tasks in total.



HADOOP = HDFS + YARN

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

DataNode and NodeManager processes are collocated on the same nodes to enable data locality. This design enables performing computations on the machines that store the data, thus minimizing the necessity of sending large chunks of data over the network, which leads to faster execution times.



YARN APPLICATIONS

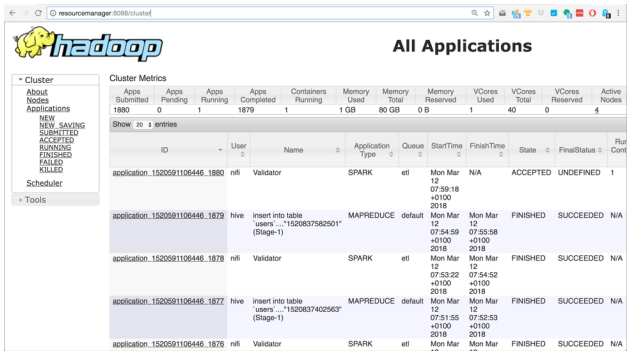
YARN is merely a resource manager that knows how to allocate distributed compute resources to various applications running on a Hadoop cluster. In other words, YARN itself does not provide any processing logic that can analyze data in HDFS. Hence, various processing frameworks must be integrated with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

Below is a list of short descriptions of the most popular distributed computation frameworks that can run on a Hadoop cluster powered by YARN.

- **MapReduce** — the traditional and the oldest processing framework for Hadoop that expresses computation as a series of map and reduce tasks. It is currently being superseded by much faster engines like Spark or Flink.
- **Apache Spark** — a fast and general engine for large-scale data processing that optimizes the computation by caching data in memory (more details in the latter sections).
- **Apache Flink** — a high-throughput, low-latency batch and stream processing engine. It stands out for its robust ability to process large data streams in real time. You can find out about differences between Spark and Flink in this comprehensive article: dzone.com/articles/apache-hadoop-vs-apache-spark
- **Apache Tez** — an engine designed with the aim of speeding up the execution of SQL queries with Hive. It is available on the Hortonworks Data Platform, where it replaces MapReduce as an execution engine for Hive.

MONITORING YARN APPLICATIONS

The execution of all applications running on the Hadoop cluster can be tracked with the ResourceManager WebUI, which is, by default, exposed on port 8088.



For each application, you can read a bunch of important information.

HOT TIP

With the ResourceManager WebUI, you can check the total amount of RAM and number of CPU cores available for processing as well as the current Hadoop cluster load. Check out "Cluster Metrics" on the top of the page.

If you click on entries in the "ID" column, you'll get more detailed metrics and statistics concerning execution of the selected application.

PROCESSING DATA ON HADOOP

There are a number of frameworks that make the process of implementing distributed applications on Hadoop easy. In this section, we focus on the most popular ones: Hive and Spark.

HIVE

Hive enables working with data on HDFS using the familiar SQL dialect.

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 working with relational databases.

Hive is not an independent execution engine. Each Hive query is translated into code in either MapReduce, Tez, or Spark (work in progress) that is subsequently executed on a Hadoop cluster.

HIVE EXAMPLE

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

```
"Creep" Radiohead piotr 2017-07-20
"Desert Rose" Sting adam 2017-07-14
"Desert Rose" Sting piotr 2017-06-10
"Karma Police" Radiohead adam 2017-07-23
"Everybody" Madonna piotr 2017-07-01
"Stupid Car" Radiohead adam 2017-07-18
"All This Time" Sting adam 2017-07-13
```

We use Hive to find the two most popular artists in July, 2017.

Upload the songs.txt file on HDFS. You can do it with the help of the "File Browser" in HUE or type the following commands using the command line tool:

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

Enter Hive using the Beeline client. You have to provide an address to HiveServer2, which is a process that enables remote clients (such as Beeline) to execute Hive queries and retrieve results.

```
# beeline
beeline> !connect jdbc:hive2://localhost:10000
<user><password>
```

Create a table in Hive that points to our data on HDFS (note that we need to specify the proper delimiter and location of the file so that Hive can represent the raw data as a table):

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

HOT TIP

After you start a session with Beeline, all the tables that you create go under the "default" database. You can change it either by providing a specific database name as a prefix to the table name or by typing the "use <database_name>;" command.

Check if the table was created successfully: `beeline> SHOW tables;`

Run a query that finds the two most popular artists in July, 2017:

```
SELECT artist, COUNT(\*) AS total
FROM songs
WHERE year(date) = 2017 AND month(date) = 7
GROUP BY artist
ORDER BY total DESC
LIMIT 2;
```

You can monitor the execution of your query with the ResourceManager WebUI. Depending on your configuration, you will see either MapReduce jobs or a Spark application running on the cluster.

Note: You can also write and execute Hive queries from HUE. There is a Query Editor dedicated for Hive with handy features like syntax auto-completion and coloring, the option to save queries, and basic visualization of the results in the form of line, bar, or pie charts.

SPARK

Apache Spark is a general purpose distributed computing framework. It is well integrated with the Hadoop ecosystem and Spark applications can be easily run on YARN.

Compared to the MapReduce - the traditional Hadoop computing paradigm - Spark offers excellent performance, ease of use, and versatility when it comes to different data processing needs.

Spark's speed comes mainly from its ability to store data in RAM between subsequent execution steps and optimizations in the execution plan and data serialization.

Let's jump straight into the code to get a taste of Spark. We can choose from Scala, Java, Python, SQL, or R APIs. Our examples are in Python. To start Spark Python shell (called pyspark), type `# pyspark.`

After a while, you'll see a Spark prompt. It means that a Spark application was started on YARN (you can go to the ResourceManager WebUI for confirmation; look for a running application named "PySparkShell").

HOT TIP

If you don't like to work with shell, you should check out web-based notebooks such as Jupyter (jupyter.org) or Zeppelin (zeppelin.apache.org).

As an example of working with Spark's Python dataframe API, we implement the same logic as with Hive, e.g. finding the two most popular artists in July, 2017.

First, we have to read in our dataset. Spark can read data directly from Hive tables: `# songs = spark.table("songs")`

Data with schema in Spark is represented as a so called dataframe. Dataframes are immutable and are created by reading data from different source systems or by applying transformations on other dataframes.

To preview the content of any dataframe, call the `show()` method:

```
# songs.show()
+-------------+---------+-----+----------+
|        title|   artist| user|      date|
+-------------+---------+-----+----------+
|        Creep|Radiohead|piotr|2017-07-20|
|  Desert Rose|    Sting| adam|2017-07-14|
|  Desert Rose|    Sting|piotr|2017-06-10|
| Karma Police|Radiohead| adam|2017-07-23|
|    Everybody|  Madonna|piotr|2017-07-01|
|   Stupid Car|Radiohead| adam|2017-07-18|
|All This Time|    Sting| adam|2017-07-13|
+-------------+---------+-----+----------+
```

To achieve the desired result, we need to use a couple of intuitive functions chained together:

```
# from pyspark.sql.functions import desc
# songs.filter("year(date) = 2017 AND month(date) = 7") \
       .groupBy("artist") \
       .count() \
       .sort(desc("count")) \
       .limit(2) \
       .show()
```

Spark's dataframe transformations look similar to SQL operators, so they are quite easy to use and understand.

HOT TIP

If you perform multiple transformations on the same dataframe (e.g. when you explore a new dataset) you can tell Spark to cache it in memory by calling the cache() method on the dataframe (e.g. songs.cache()). Spark will then keep your data in RAM and avoid hitting the disk when you run subsequent queries, giving you an order of magnitude better performance.

Dataframes are just one of the available APIs in Spark. There are also APIs and libraries for near real-time processing (Spark Streaming), machine learning (MLlib), or graph processing (GraphFrames).

Thanks to Spark's versatility, you can use it to solve a wide variety of your processing needs, staying within the same framework and sharing pieces of code between different contexts (e.g. batch and streaming).

Spark can directly read and write data to and from many different data stores, not only HDFS. You can easily create dataframes from records in a table in MySQL or Oracle, rows from HBase, JSON files on your local disk, index data in ElasticSearch, and many, many others.

OTHER TOOLS FROM THE HADOOP ECOSYSTEM

The Hadoop ecosystem contains many different tools to accomplish specific tasks needed by modern big data platforms. Below, you can see a list of popular and important projects that were not covered in previous sections.

Sqoop — an indispensable tool to move data in bulk from relational datastores and HDFS/Hive, and the other way around. You interact with Sqoop using the command line, selecting the desired action and providing a bunch of necessary parameters controlling the data movement process. Importing data about users from a MySQL table is as easy as typing the following command:

```
# sqoop import \
--connect jdbc:mysql://localhost/streamrock \
--username $(whoami) -P \
--table users \
--hive-import
```

Note: Sqoop uses MapReduce to transfer data back and forth between the relational datastore and Hadoop. You can track a MapReduce application submitted by Sqoop in the ResourceManager WebUI.

Oozie — a coordination and orchestration service for Hadoop. With Oozie, you can build a workflow of different actions that you want to perform on a Hadoop cluster (e.g. HDFS commands, Spark applications, Hive queries, Sqoop imports, etc.) and then schedule a workflow for automated execution.

HBase — a NoSQL database built on top of HDFS. It enables very fast random reads and writes of individual records using row keys.

Zookeeper — a distributed synchronization and configuration management service for Hadoop. A number of Hadoop services take advantage of Zookeeper to work correctly and efficiently in a distributed environment.

SUMMARY

Apache Hadoop is the most popular and widely-used platform for big data processing, thanks to features like linear scalability, high-level APIs, the ability to run on heterogeneous hardware (both on-premise and in the cloud), fault tolerance, and an open-source nature. Hadoop has been successfully deployed in production by many companies for over a decade.

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 cheaply store and process huge amounts of data in a distributed and highly scalable way.

ADDITIONAL RESOURCES

- hadoop.apache.org
- hive.apache.org
- spark.apache.org
- spark.apache.org/docs/latest/sql-programming-guide.html
- dzone.com/articles/apache-hadoop-vs-apache-spark
- dzone.com/articles/hadoop-and-spark-synergy-is-real
- sqoop.apache.org
- dzone.com/articles/sqoop-import-data-from-mysql-to-hive
- oozie.apache.org
- tez.apache.org

Major packaged distributions:

- Cloudera: cloudera.com/content/cloudera/en/products-and-services/cdh.html
- MapR: mapr.com/products/mapr-editions
- Hortonworks: hortonworks.com/hadoop/

Written by Piotr Krewski, Founder and Big Data Consultant at GetInData

Piotr 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, where he helps companies with building scalable, distributed architectures for storing and processing big data. Piotr serves also as a Hadoop Instructor, delivering GetInData proprietary trainings for administrators, developers, and analysts working with big data solutions.

Written by Adam Kawa, CEO and Founder at GetInData

Adam became a fan of big data after implementing his first Hadoop job in 2010. Since then, he has been working with big data at Spotify (where he had proudly operated one of the largest and fastest-growing Hadoop clusters in Europe), Truecaller, the University of Warsaw, Cloudera Training Partner, and more. Three years ago, he co-founded GetInData: a company that helps its customers become data-driven and that builds innovative big data solutions. Adam is also a blogger, a co-organizer of Warsaw Hadoop User Group, and a frequent speaker at major big data conferences and meetups.

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