

Big Data for Engineers – Exercises

Spring 2020 – Week 3 – ETH Zurich

Introduction

This week we will cover mostly theoretical aspects of Hadoop and HDFS and we will discuss advantages and limitations of different storage models. The mandatory reading is [1] [Shvachko, K. et al. \(2010\). The Hadoop Distributed File System. In MSST.](#)

What is Hadoop?

"Hadoop provides a **distributed file system** and a **framework for the analysis and transformation** of very **large** data sets using the **MapReduce paradigm**." [1]

Several components are part of this framework. In this course you will study HDFS, MapReduce and HBase while this exercise focuses on HDFS and storage models.

Component	Description	First developer
HDFS	Distributed file system	Yahoo!
MapReduce	Distributed computation framework	Yahoo!
HBase	Column-oriented table service	Powerset (Microsoft)
Pig	Dataflow language and parallel execution framework	Yahoo!
Hive	Data warehouse infrastructure	Facebook
ZooKeeper	Distributed coordination service	Yahoo!
Chukwa	System for collecting management data	Yahoo!
Avro	Data serialization system	Yahoo! + Cloudera

1. The Hadoop Distributed File System

1.1 – State which of the following statements are true:

1. The HDFS namespace is a hierarchy of files and directories.
2. In HDFS, each block of the file is either 64 or 128 megabytes depending on the version and distribution of Hadoop in use, and this cannot be changed.
3. A client wanting to write a file into HDFS, first contacts the NameNode, then sends the data to it. The NameNode will write the data into multiple DataNodes in a pipelined fashion.
4. A DataNode may execute multiple application tasks for different clients concurrently.
5. The cluster can have thousands of DataNodes and tens of thousands of HDFS clients per cluster.
6. HDFS NameNodes keep the namespace in RAM.
7. The locations of block replicas are part of the persistent checkpoint that the NameNode stores in its native file system. ([more detail in \[1\]](#))
8. If the block size is set to 64 megabytes, storing a file of 80 megabytes will actually require 128 megabytes of physical memory (2 blocks of 64 megabytes each).
9. Every 6 hours the *NameNode* asks the *DataNode* for a complete report on the blocks stored on it.
10. HDFS is optimised for latency.

1.2 – Where is the information stored? For each of the following, say if the information is stored in the disk of the NameNode, in the disk of a DataNode, in disks of both of them, or in none of them.

1. the list of blocks belonging to each file [*NameNode* | *DataNode*]
2. the files containing the actual blocks of data [*NameNode* | *DataNode*]
3. the block's metadata including checksum and generation stamp [*NameNode* | *DataNode*]
4. the location of block replicas [*NameNode* | *DataNode*] ([more detail in \[1\]](#))

1.3 – A typical filesystem block size is 4096 bytes. How large is a block in HDFS? List at least two advantages of such choice.

1.4 – How does the hardware cost grow as function of the amount of data we need to store in a Distributed File System such as HDFS? Why?

1.5 – Scalability, Durability and Performance on HDFS

Explain how HDFS accomplishes the following requirements:

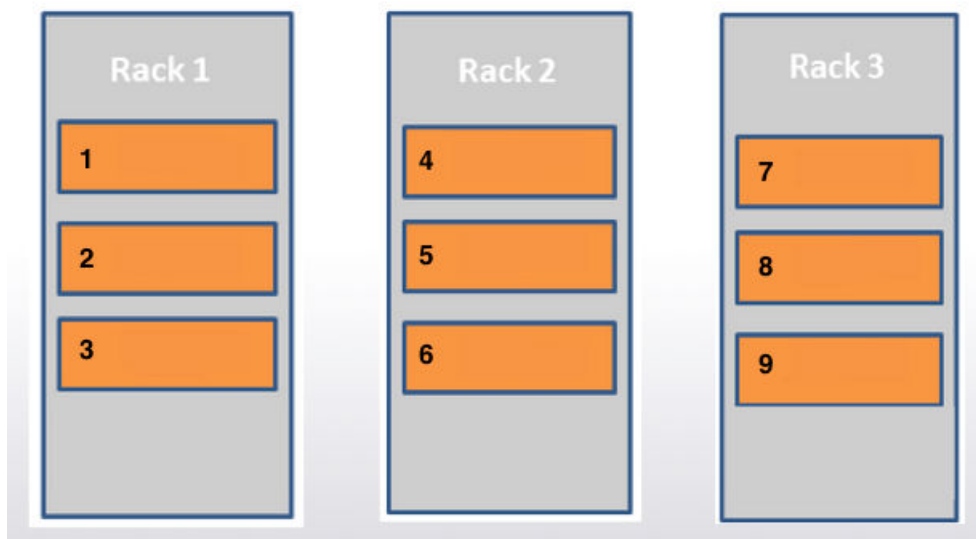
1. Scalability
2. Durability
3. High sequential read/write performance

2. File I/O operations and replica management.

2.1 – Replication policy

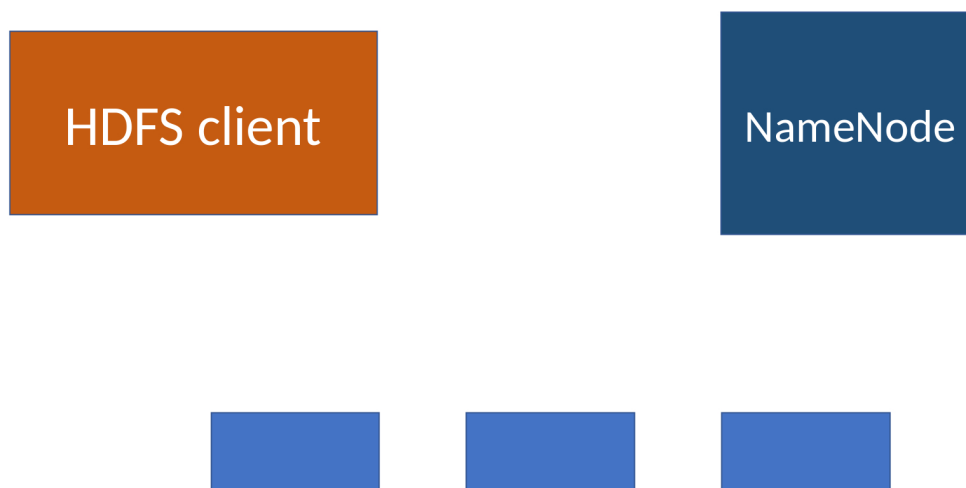
Assume your HDFS cluster is made of 3 racks, each containing 3 DataNodes. Assume also the HDFS is configured to use a block size of 100 megabytes and that a client is connecting from outside the datacenter (therefore no DataNode is privileged).

1. The client uploads a file of 150 megabytes. Draw in the picture below a possible blocks configuration according to the default HDFS replica policy. How many replicas are there for each block? Where are these replicas stored?
2. Can you find a with a different policy that, using the same number of replicas, improves the expected availability of a block? Does your solution show any drawbacks?
3. Referring to the picture below, assume a block is stored in Node 3, as well as in Node 4 and Node 5. If this block of data has to be processed by a task running on Node 6, which of the three replicas will be actually read by Node 6?



2.2 – File read and write data flow.

To get an idea of how data flows between the client interacting with HDFS, consider a diagram below which shows main components of HDFS.



DataNode

DataNode

DataNode

1. Draw the main sequence of events when a client copies a file to HDFS.
2. Draw the main sequence of events when a client reads a file from HDFS.
3. Why do you think a client writes data directly to datanodes instead of sending it through the namenode?

3. Storage models

Last week we delve into the key-value model (incl. object storage, e.g., Amazon S3/Azure Blob). This week, we can contrast the object storage in the key-value model with block storage, which is used for distributed file systems.

3.1 – List two differences between Object Storage and Block Storage.

3.2 – Compare Object Storage and Block Storage. For each of the following use cases, say which technology (object or block storage) better fits the requirements and briefly justify why.

1. Store Netflix movie files in such a way they are accessible from many client applications at the same time [Object storage | Block Storage]
2. Store experimental and simulation data from CERN [Object storage | Block Storage]
3. Store the auto-backups of iPhone/Android devices [Object storage | Block Storage]

3.3 – Cost of Object Storage (Optional)

Azure Object Storage offers different access tiers, which allow you to store blob object data in the most cost-effective manner.

Imagine you want to have a copy of your important documents, photos and videos to both ensure durability and to share them across your several devices. You need about 600GB of storage space. Two possibilities would be [hot and cool storage tiers on Azure Blob Storage](#).

1. Explain the difference between hot and cool storages.
2. Compare costs of cool and hot storages on [Azure Storage](#). Explain why prices are different. Which one would you choose? Why?
3. Another possible solution would be to pay for a [Dropbox Pro account](#). Compare costs of Dropbox and Azure storage. Which one is cheaper?
4. What about buying an external hard drive? List two advantages and two disadvantages of this solution.

Explore on your own

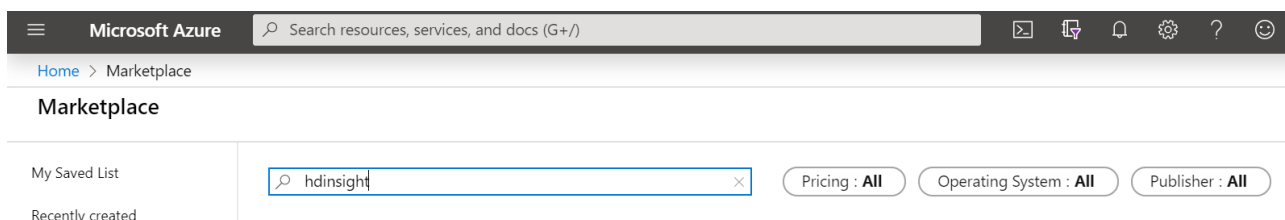
4. Run an HDFS cluster

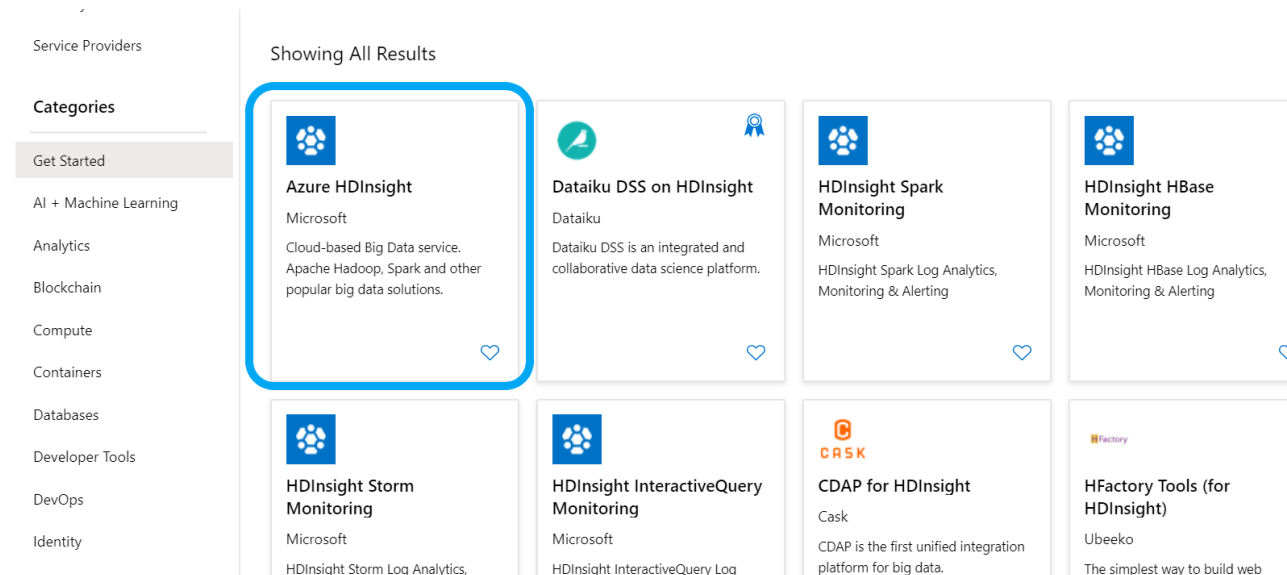
It's time to work on a real environment! You will now create an HDFS cluster running on Azure and fill it with data. The purpose of this exercise is to familiarise you with shell commands and the operations to administer an HDFS deployment.

4.1 – Do the following to set up an HDFS cluster:

Important: we want you to use a small but real cluster for running HDFS rather than a single machine. But, these clusters burn Azure credit very quickly—the cheapest configuration consumes roughly **1 CHF per hour**, which is a lot relative to your overall credit—so it is very important for you to ****delete**** your cluster once you are done. Luckily, *it is possible to keep your data intact when you delete a cluster*, and see it again when you recreate it; we will touch upon this in the process. Now, let's start ...

1. Open the [Azure portal](#) and click on the "+ Create a Resource" button on the left. Type "hdinsight" in the search box, and select "HDInsight". HDInsight is Microsoft's cloud service which wraps Hadoop, HBase, Spark and other Big Data technologies; read more [here](#).





1. Click on Create, fill in the form with cluster name, user names and passwords, and select "Hadoop" as the cluster type. Create a new resource group, e.g., "exercise03". Click "Next".

1. In the next we need to configure the location to store all the cluster data. The canonical storage layer for an HDFS cluster uses the Blob service of Windows Azure Storage and the primary advantage is that it allows you to delete your HDFS cluster without losing the stored data: you can recreate the cluster using the same Azure Storage Account and the same container and you will see the same data. This is useful, for example, if you don't have time to finish this exercise in one sitting: you can just delete your cluster, recreate it later, and continue your work. Azure storage is selected by default (see the screenshot). In "Select a Storage Account" click "Create new" and specify a name. **Important: if you are recreating your HDFS cluster and want to see the existing data, then choose "Select existing" and set the container name to the one that you see in the "Storage Accounts" tab of Azure—by default Azure generates a new container name every time you create a cluster, which then points to a different container.** Leave everything else as it is and click "Next".
2. In the "Security + networking" step do not choose anything and just click "Next".
3. Now we need to choose the configuration of the nodes in our HDFS cluster. Nodes in HDInsight have similar names yet types, compared with the concepts of HDFS you have learned from the lecture. In this exercise, you will see *head node* (aka *namenode*) and *worker node* (aka *datanode*). To read more, this blog on "[Nodes in HDInsight](#)" is helpful. It will be enough to have only 2 **worker** nodes (see the screenshot). As for the node size, let us be wise and select an economical option: from the drop-down menu choose the option "**A4 - v2**"(see the screenshot); do the same for the **Head** nodes. Click "Next".

Create HDInsight cluster

Node configuration

Configure your cluster's size and performance, and view estimated cost information.

The cost estimate represented in the table does not include subscription discounts or costs related to storage, networking, or data transfer.

i This configuration will use 16 of 50 available cores in the East US region.
[View cores usage](#)

[+ Add application](#)

Node type	Node size	Number of ...	Estimated co
Head node	A4 v2 (4 Cores, 8 GB RAM), 0.22 EUR/hour	2	0.44 EUR
Worker node	A4 v2 (4 Cores, 8 GB RAM), 0.22 EUR/hour	2	0.44 EUR

☐ Enable autoscale (Preview) [Learn](#) [More](#)

[Review + create](#)

[« Previous](#)

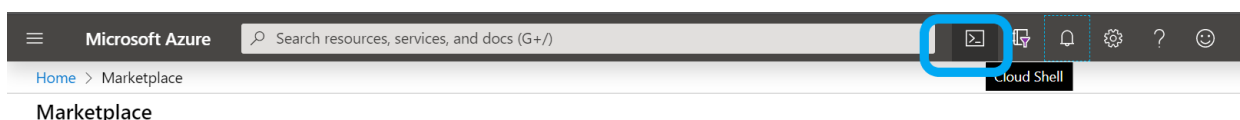
[Next: Tags »](#)

4. All the fields on the "Tags" step can be left as they are. Simply proceed by clicking "Next".
5. In the last step, "Review + create", check if the settings are as you intend. These clusters are expensive, so it is worth checking the price estimate at this step: for me it is 0.87 CHF/hour; if your price is larger than this, check your node sizes and counts. When done, initiate the cluster creation by clicking "Create". The process will take time, around 15–25 minutes; in my own case it took 20 minutes.

4.2 – Accessing your cluster

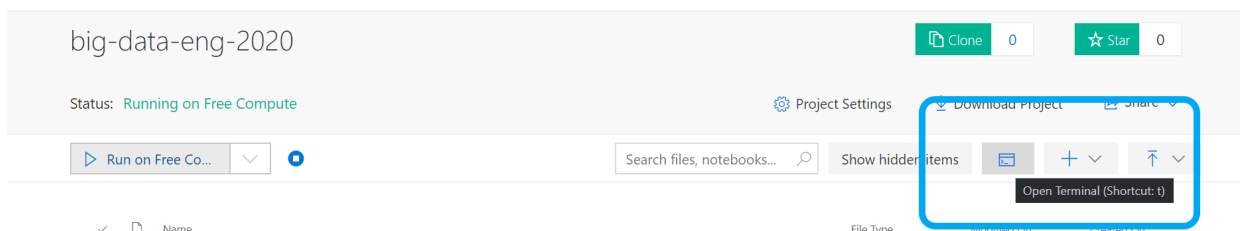
The standard way to interact with an HDFS cluster is via the Java API or a command-line interface. We will use the latter and for this, you will need to run the `ssh` program in a terminal in order to connect to your cluster. There are three options of how you can do this:

1. **On your own machine** you can just use a normal terminal if you have `ssh` installed. Linux usually has it, as does MacOS. Windows doesn't have it by default (powershell on Win10 does, though), but Windows users can use one of the browser-based options, which are described next, and the other option is to install [PuTTY](#).
2. **In your browser, two possibilities:**
 - A. Use the **Azure Cloud Shell**. Click on the Cloud Shell icon at the top of Azure Dashboard toolbar:



Choose "bash". It will request your approval for creating a Storage Account required for the shell; choose the corresponding subscription and you can also create a new storage account or use an old storage account.

- B. Use the built-in **terminal on Jupyter**. In your [notebooks.azure.com](#), find the terminal button on the right side, click on it.



In your terminal of choice, run the following (this command with everything filled-in is also available on the Azure page of your HDFS cluster, if you click "SSH + Cluster login" and select the host name):

```
ssh <ssh_user_name>@<cluster_name>-ssh.azurehdinsight.net
```

In this command, `<ssh_user_name>` is the "ssh username" that you have chosen in the first step of creating the HDFS cluster, and `<cluster_name>` also comes from that form. Note that the cluster name has to be suffixed with `-ssh`.

If after running the `ssh` command you see a message similar to this:

```
Welcome to HDInsight.
```

[...]

To run a command as administrator (user "root"), use "sudo <command>".
See "man sudo_root" for details.

```
<ssh_user_name>@hn0-cluster:~$
```

then you have successfully connected to your HDFS cluster. Now proceed to the next task.

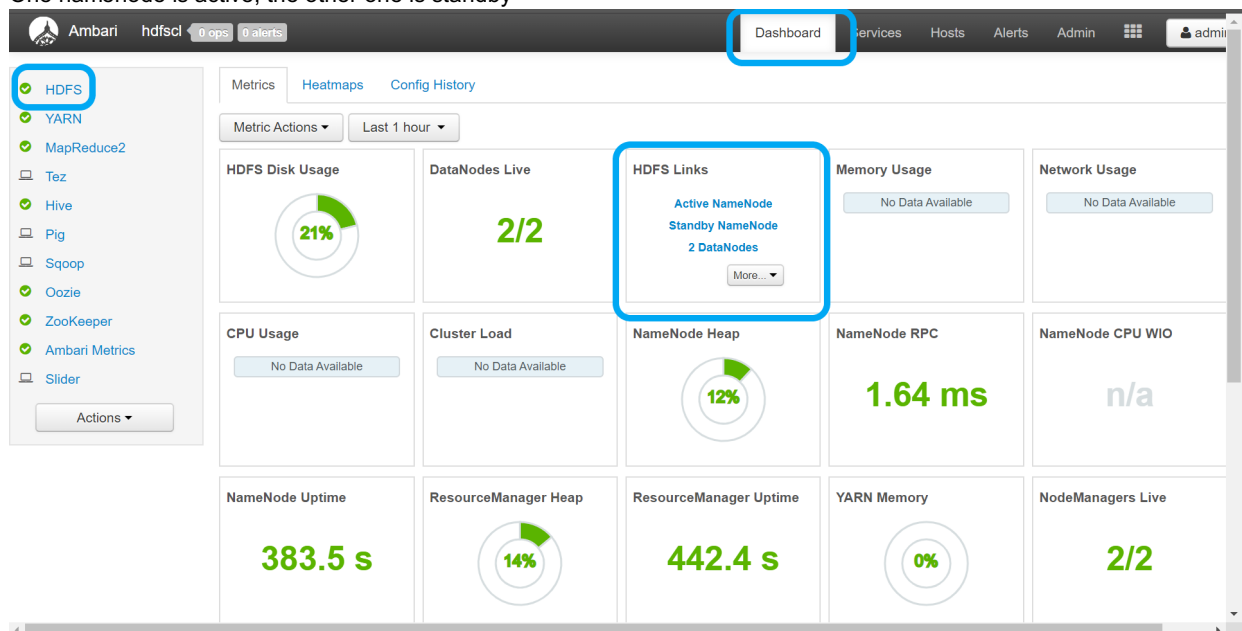
4.3 – Monitor the HDFS cluster

In the above screenshot, we highlight the **Cluster Dashboards** powered by Ambari. The Apache Ambari project is aimed at making Hadoop management simpler by developing software for provisioning, managing, and monitoring Apache Hadoop clusters. Ambari provides an intuitive, easy-to-use Hadoop management web UI backed by its RESTful APIs ([source](#)).

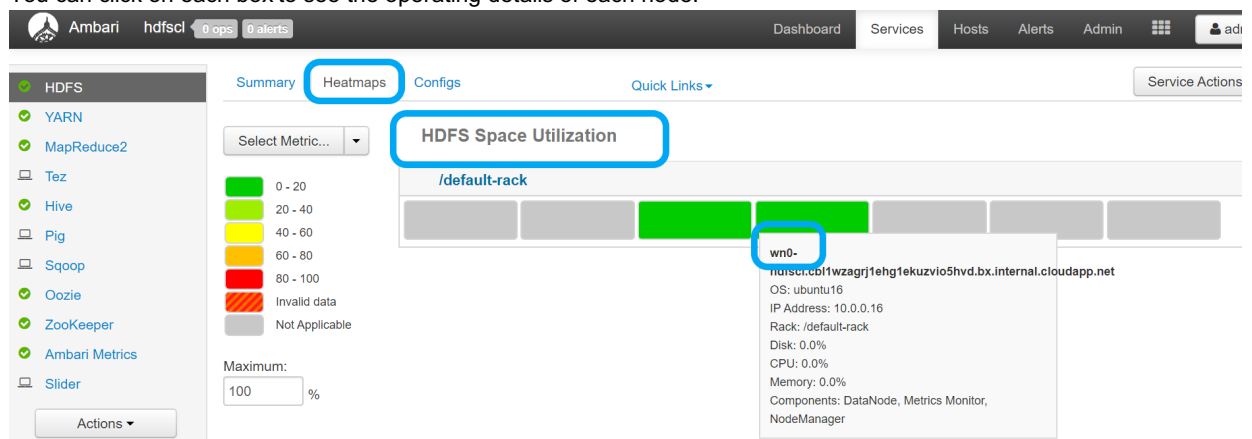
We utilize this monitoring dashboard to inspect the behaviors of our HDFS cluster. You will see prominent behavior changes (e.g., resource usage live for the nodes and network usage going up) esp. when uploading some new data.

Click into the Ambari home. You can explore the following:

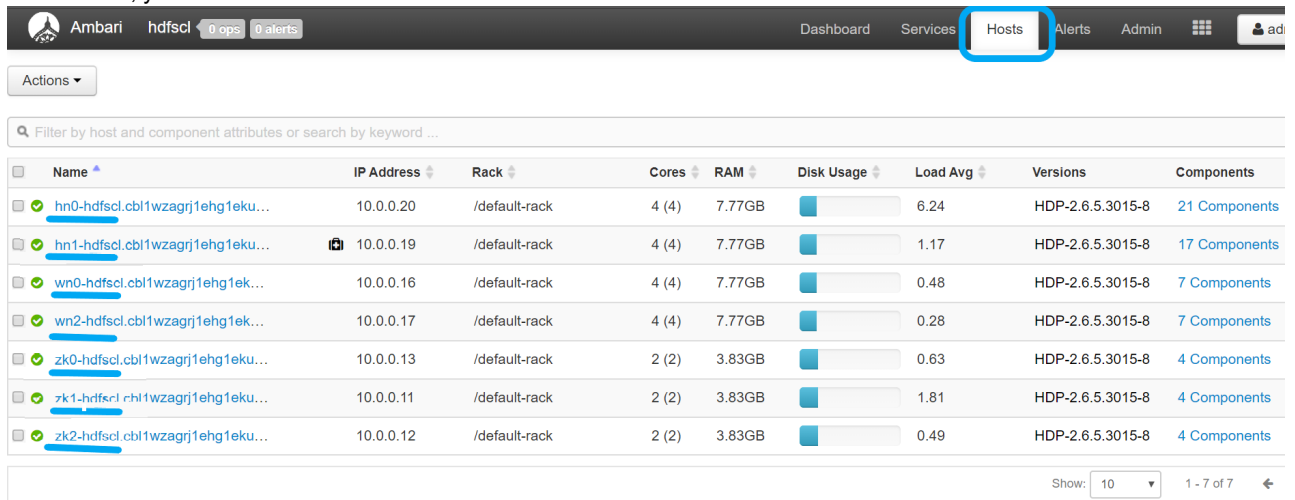
1. An overview of the cluster (namenode, datanode)
 - One namenode is active, the other one is standby



1. Space utilization on HDFS and the node details (2 namenodes, 2 datanodes, 3 zookeeper nodes)
 - The "Zookeeper" nodes are selected by default (Zookeeper is a [distributed coordination service](#) used by HDFS). By default, HDInsight provides 3 Zookeeper nodes.
 - You can click on each box to see the operating details of each node.



1. Under Hosts, you will find more details



Name	IP Address	Rack	Cores	RAM	Disk Usage	Load Avg	Versions	Components
hn0-hdfscl.cbl1wzagrijehg1eku...	10.0.0.20	/default-rack	4 (4)	7.77GB		6.24	HDP-2.6.5.3015-8	21 Components
hn1-hdfscl.cbl1wzagrijehg1eku...	10.0.0.19	/default-rack	4 (4)	7.77GB		1.17	HDP-2.6.5.3015-8	17 Components
wn0-hdfscl.cbl1wzagrijehg1eku...	10.0.0.16	/default-rack	4 (4)	7.77GB		0.48	HDP-2.6.5.3015-8	7 Components
wn2-hdfscl.cbl1wzagrijehg1eku...	10.0.0.17	/default-rack	4 (4)	7.77GB		0.28	HDP-2.6.5.3015-8	7 Components
zk0-hdfscl.cbl1wzagrijehg1eku...	10.0.0.13	/default-rack	2 (2)	3.83GB		0.63	HDP-2.6.5.3015-8	4 Components
zk1-hdfscl.cbl1wzagrijehg1eku...	10.0.0.11	/default-rack	2 (2)	3.83GB		1.81	HDP-2.6.5.3015-8	4 Components
zk2-hdfscl.cbl1wzagrijehg1eku...	10.0.0.12	/default-rack	2 (2)	3.83GB		0.49	HDP-2.6.5.3015-8	4 Components

4.4 – Upload a file into HDFS

Let us go over some common [commands](#):

```
ls, cat, mkdir, cp, rm, rmdir, copyFromLocal, copyToLocal
```

A quick note: we will use the `hadoop fs` keywords as a prefix to the commands, but we could have used `hdfs dfs` keywords as well. The only difference is that `hadoop fs` is a more “generic” command that allows you to interact with multiple file systems including Hadoop (but also your local file system), whereas `hdfs dfs` is specific to HDFS.

1. List the directories or files: `ls` (some of these commands could be slow)

- For a file `ls` returns statistics on the file: `$hadoop fs -ls /example/data/fruits.txt`
This gives you the following: `-rw-r--r-- 1 root supergroup 66 2020-03-01 17:42 /example/data/fruits.txt`
- For a directory it returns list of its direct children as in Unix: `$hadoop fs -ls /example`
This gives you the following:
`Found 2 items`
`drwxr-xr-x - root supergroup 0 2020-03-01 17:43 /example/data`
`drwxr-xr-x - root supergroup 0 2020-03-01 17:43 /example/jars`
- List files in a directory: `$hadoop fs -ls /example/data/*`
- List all the directories in your HDFS: `$hadoop fs -ls -R /`

2. Copies source paths to stdout: `cat`

- Inspect the file content (**DO NOT** do it for a large file):

```
$hadoop fs -cat /example/data/fruits.txt
```

3. Copy files from source to destination: `cp`

```
$hadoop fs -cp /example/data/fruits.txt /example/data/fruits-copy.txt
```

 This shouldn't return anything.

4. Delete files: `rm`

```
$hadoop fss -rm /example/data/fruits-copy.txt
```

This gives you: Deleted /example/data/fruits-copy.txt

- When you then check `$hadoop fs -ls /example/data/fruits-copy.txt`, you will see the file does not exist anymore by getting this message: `ls: '/example/data/fruits-copy.txt': No such file or directory`

5. Creates directories: `mkdir`

- We create a folder for our exercise by `$hadoop fs -mkdir /ex03`

6. Copy a file from the file system where your terminal locates to HDFS: `copyFromLocal`

- First we create a file on the file system where your terminal locates (can be local or on a cluster):

```
$ echo "we create a file on the cluster(hn0) file system" > cluster.txt
```

```
$ cat cluster.txt
```

- These two commands are equivalent:

```
$hadoop fs -copyFromLocal -f cluster.txt /ex03/
```

```
$hadoop fs -put -f cluster.txt /ex03/
```

- The `-f` option will overwrite the destination if it already exists.

7. The reverse operation of `copyFromLocal`, i.e., download a file from HDFS to the file system where your terminal locates: `copyToLocal`

- `$hadoop fs -copyToLocal /example/data/fruits.txt`
- If you do `$ ls` in your terminal, you should see the "fruits" file there.

8. Delete a directory: `rmdir`

- We cannot delete a non-empty folder directly, so we need to remove the files inside by `$hadoop fss -rm /ex03/*`
- We then remove the folder `$hadoop fs -rmdir /ex03`
- If you do `ls` again on the deleted folder, it should not exist any more. `$hadoop fs -ls /ex03`

Now we try to upload a big file (>1G) to our HDFS cluster and you can inspect the cluster with Ambari.

Download a compressed dump of all the articles on Wikipedia: `$ wget`

```
https://bigdataforeng.blob.core.windows.net/ex05/wiki.tar.gz
```

Next, let's upload this file from the local file system to HDFS:

```
$hadoop fs -mkdir /bigdata
```

```
$hadoop fs -put ~/wikibig.tar.gz /bigdata
```

```
$hadoop fs -ls /bigdata
```

You should see a listing of the following form with details about the uploaded file (permissions, ownership, size and modification date):

```
Found 1 items
```

```
-rw-r--r--  1 sshuser supergroup 1116569600 2020-03-01 17:45 /bigdata/wikibig.tar.gz
```

This was a quick overview of the basic shell commands and in a following exercise we will run a MapReduce computation over data stored in HDFS.

Feel free to try out the commands on your own. For instance, pick an image file in your computer (or you can also download a random one) and try to upload it to HDFS. You may need to create an empty directory before uploading.

Important: Don't forget to terminate and delete your cluster once you are done (!)