Big Data – Exercises

Fall 2021 – 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.

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.

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.
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).

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

1.3 – 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?

1.4 – Single Point of Failure

1. Which component is the main single point of failure in Hadoop?
2. What is the Secondary NameNode?

1.5 – Scalability, Durability and Performance on HDFS

Explain how HDFS accomplishes the following requirements:

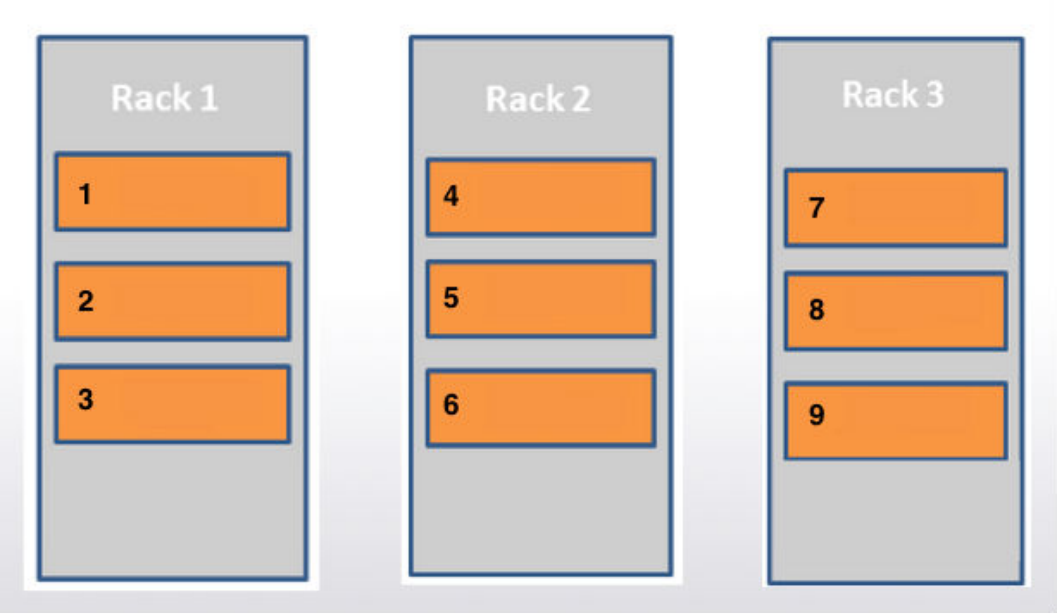
1. Scalability
2. Durability
3. High sequential read/write performance

2. File I/O operations and replica management.

2.1 – Replication policy

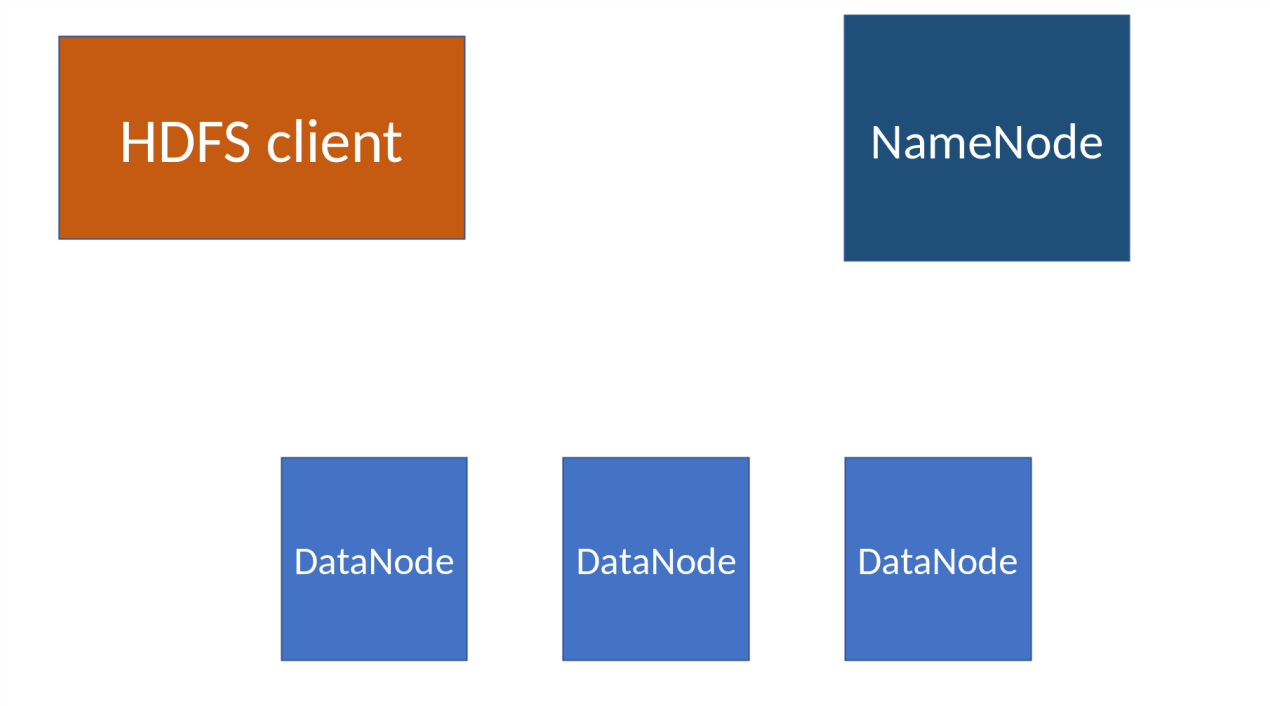
Assume your HDFS cluster is made of 3 racks, each containing 3 DataNodes. Assume also the HDFS is configured to use a block size of 100 megabytes and that a client is connecting from outside the datacenter (therefore no DataNode is priviledged).

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.

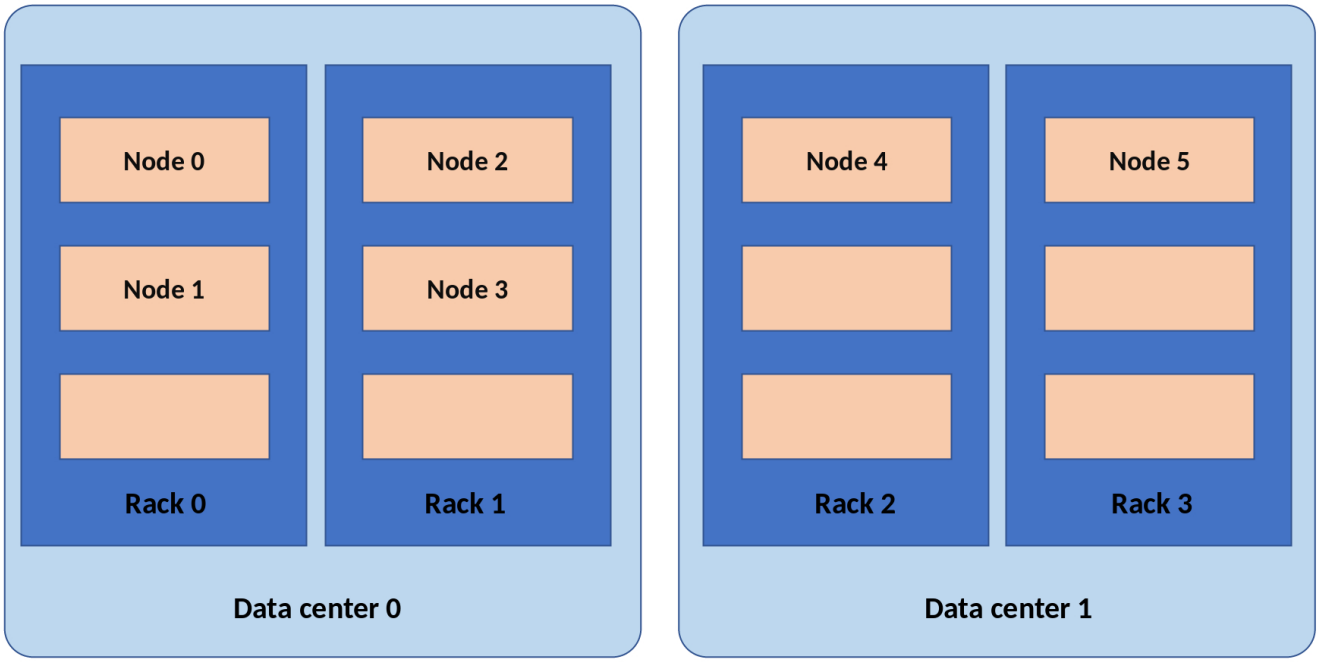
To get an idea of how data flows between the client interacting with HDFS, consider a diagram below which shows main components of HDFS.



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?

2.3 – Network topology.

HDFS estimates the network bandwidth between two nodes by their distance. The distance from a node to its parent node is assumed to be one. A distance between two nodes can be calculated by summing up thier distances to their closest common ancestor. A shorter distance between two nodes means that the greater bandwidth they can utilize to transfer data. Consider a diagram of a possible hadoop cluster over two datacenters below.



Calculate following distances using the distance rule explained above:

1. Node 0 and Node 1
2. Node 0 and Node 2
3. Node 1 and Node 4
4. Node 4 and Node 5
5. Node 2 and Node 3
6. Two processes of Node 1

3. Storage models

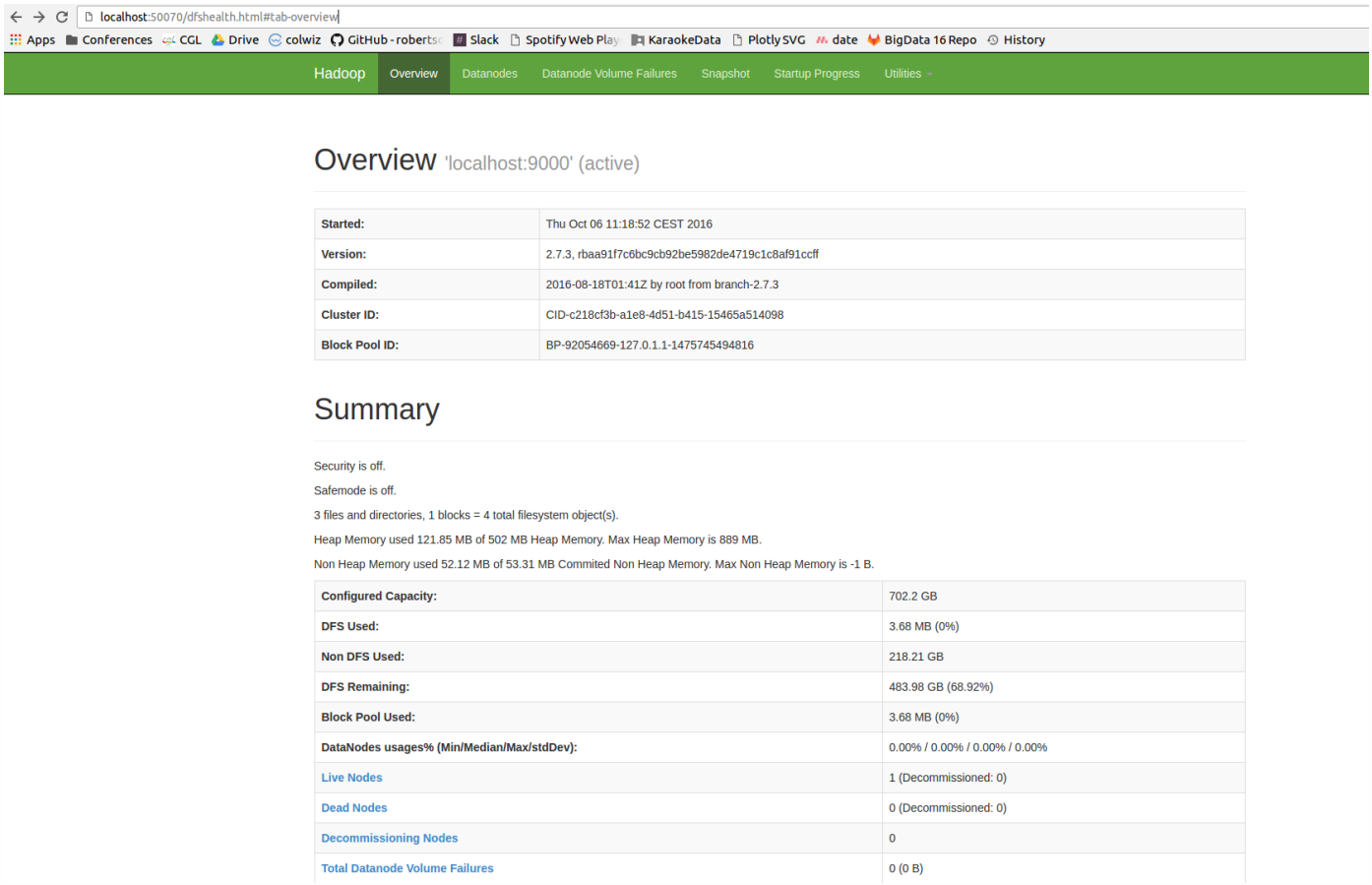
3.1 – List two differences between Object Storage and Block Storage.

3.2 – Compare Object Storage and Block Storage. For each of the following use cases, say which technology better fits the requirements.

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*]

4. Working Docker-Hadoop

Build and run the Hadoop docker image by `docker-compose up -d` in the `docker-hadoop` directory. If completed successfully, you should be able to browse `http://localhost:9870/` and visualize the web interface of the daemon which should look similar to the following image.
In the `Datanodes` tab you should see a single operating datanode.

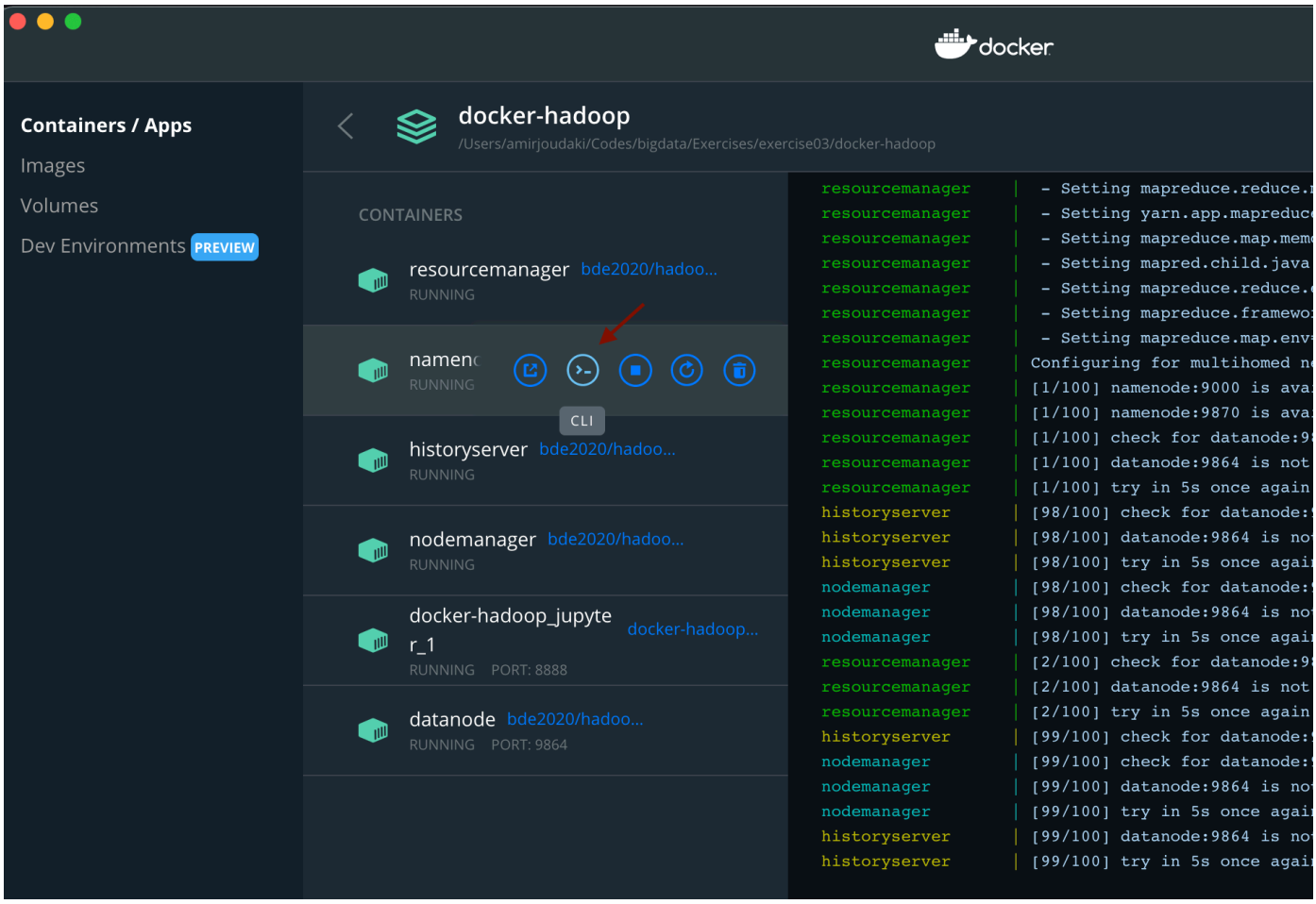


Connecting to containers

Each Hadoop cluster is set up in one of the three supported modes:

- Local (Standalone) Mode
- Pseudo-Distributed Mode
- Fully-Distributed Mode

By default Hadoop runs in Local Mode but we will run it in the *Pseudo-Distributed Mode*. This will allow you to run Hadoop on a single-node (your computer) simulating a distributed file system, with datanode and namenode running in separate containers. For this excercise you will only need to connect to `namenode` and `datanode` containers. To connect to namenode container can either use the Docker dashboard interface by navigating to `docker-hadoop` app, and selecting `CLI` option from the `namenode` container (see image below).



Atlernatively, you can run `docker exec -it namenode /bin/bash` in terminal. To connect to a datanode you can similarly find it in the dashboard or run `docker exec -it namenode /bin/bash` in the terminal.
Both approaches will give you shell access on the corresponding container.

4.1 – Upload a file into HDFS

Connect to the namenode by `docker exec -it namenode /bin/bash`

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. (check here for help)

1. Which command do you use to upload from the local file system to HDFS?
2. Which information can you find if you use `Utilities -> Browse the file system` in the daemon web interface?

4.3 – Local File System

1. Download a text file using `curl` command

```
curl https://fengche.co/robots.txt -o robots.txt
```

Use HDFS commands to create a directory, copy the text file from your local file system to HDFS. Use `cat` to check if the file is the same on the local and distributed systems.

Hint: you may use the following HDFS commands `-mkdir` for directory, `-copyFromLocal` for uploading the file, and `-cat` for printing them

2. Try to locate the file on a datanode. To connect to a datanode by running
   `docker exec -it datanode /bin/bash`
   This will give you shell access to the data node machine. cd into `/hadoop/dfs/data/current/` directory and follow the directories until there are only files. Can you check if the file contents are the same as the one you uploaded? Use `ls -l` to check the size of the file size on the local
3. Now try to upload a file to HDFS that is ~150MB. On Unix-based system you can also generate such a file filled with zero using:

```
dd if=/dev/zero of=zeros.dat bs=1M count=150
```

How many blocks the file is split into?

4.4 Demystifying FsImage & Edits, & Checkpoint

When the NameNode starts up, or a checkpoint is triggered by a configurable threshold,:

- it reads the FsImage and EditLog from disk
- it applies all the transactions from the EditLog to the in-memory representation of the FsImage
- it flushes out this new version into a new FsImage on disk.
- It truncates the old EditLog because its transactions have been applied to the persistent FsImage.

A checkpoint can be triggered:

> at a given time interval (dfs.namenode.checkpoint.period) expressed in seconds,
> or after a given number of filesystem transactions have accumulated (dfs.namenode.checkpoint.txns).

If both of these properties are set, the first threshold to be reached triggers a checkpoint.

1. query the configuration file
   Query the config file

- `hdfs getconf -confKey dfs.namenode.checkpoint.period`
- `hdfs getconf -confKey dfs.namenode.checkpoint.txns`
- the fsimage & edit logs location `hdfs getconf -confKey dfs.namenode.name.dir`, I get something like

`file:///hadoop/dfs/name`
- find the fsimage and edit logs in the `current` directory. They must be named like `fsimage_0000000000000000000` & `edits_inprogress_0000000000000000001`
- output edits `hdfs oev -p xml -i /hadoop/dfs/name/current/edits_inprogress_0000000000000000001 -o edits.xml`
- output fsimage `hdfs oiv -p XML -i /hadoop/dfs/name/current/fsimage_0000000000000000000 -o fsimage.xml`

2. can you make sense of the outputs?

4.5 Changing Block Size (optional)

As explained in the tutorials, to change HDFS configurations you edit `etc/hadoop/core-site.xml` and `etc/hadoop/hdfs-site.xml`. In the docker app, you can modify the variables in the `hadoop.env`. For example the following line

```
# hadoop.env
CORE_CONF_fs_defaultFS=hdfs://namenode:9000
```

`CORE_CONF` corresponds to `core-site.xml`. and the second part `fs_defaultFS=hdfs://namenode:9000` will be transformed into:

```
<property>
    <name>fs.defaultFS</name><
    value>hdfs://namenode:9000
    </value>
</property>
```

For more details see here.

Try changing the default block size of HDFS to see its affect on read & write performance. You can change the block size by modifying the follwoing line in `hadoop.env`:
`HDFS_CONF_dfs_block_size=1048576`
The value `1048576` determines the block size in bytes, which in this case is `2^20 bytes` or 1 megabytes.

> ***NOTE:*** that for these configuration changes to take effect you must restart the docker app!

1. Create a file with size ~150MB and uploade the file to HDFS. Check number of blocks via the Web interface.
2. For each of the following block sizes 1048576, 134217728, measure the time to transfer from local to HDFS, from HDFS to local, and removing the file.
   You can run the following commands

```
time hadoop fs -copyFromLocal zeros.dat /myfolder/zeros.dat
time hadoop fs -get /myfolder/zeros.dat zeros.dat
time hadoop fs -rm /myfolder/zeros.dat
```

Can you make sense of the results?

> ***NOTE:*** make sure to remove the files before uploading them, so that no caching will distort the measurements