# Hadoop setup

## Materials

**The best materials:**

* [youtube - Data Engineering](https://www.youtube.com/watch?v=_iP2Em-5Abw) - multi node hadoop set up without docker
* [www.cloudduggu.com](https://www.cloudduggu.com/hadoop/installation-multi-node-cluster/) - multi node hadoop set up without docker
* [www.michael-noll.com](https://www.michael-noll.com/tutorials/running-hadoop-on-ubuntu-linux-multi-node-cluster/#formatting-the-hdfs-filesystem-via-the-namenode) - multi node hadoop set up without docker
* [www.confessionsofadataguy.com](https://www.confessionsofadataguy.com/create-your-very-own-apache-spark-hadoop-cluster-then-do-something-with-it/) - Setup of both Hadoop and Spark. there is no info about on which nodes (servers) execute which commands in terminal for setting up hadoop. But the part for Spark seems fine.

**Official documentation:**

* Hadoop official documentation about cluster setup: [hadoop.apache.org](https://hadoop.apache.org/docs/r2.8.0/hadoop-project-dist/hadoop-common/ClusterSetup.html)
* Hadoop official HDFS guide: [hadoop.apache.org](https://hadoop.apache.org/docs/r2.8.0/hadoop-project-dist/hadoop-hdfs/HdfsUserGuide.html#Related_Documentation)

**Other materials:**

* [medium.com](https://freedium.cfd/https:/blog.det.life/developing-multi-nodes-hadoop-spark-cluster-and-airflow-in-docker-compose-part-1-10331e1e71b3) - medium article about multi node hadoop and spark set up with docker. It creates multiple data nodes as separate docker containers but everything is running on a single machine using docker compose.
* [medium.com](https://freedium.cfd/https:/medium.com/@rubenafo/some-tips-to-run-a-multi-node-hadoop-in-docker-9c7012dd4e26) - hadoop multi node set up. There is a missing piece about generating ssh keys. Here different data nodes are created in a separate docker containers and all of them are running on a single machine. Docker compose is not used so some additional task related to networks in Docker are required.
* [linkedin.com](https://www.linkedin.com/pulse/setup-multi-node-hadoop-cluster-using-docker-komal-suthar/) - hadoop multi node set up. Some actions are done with docker, some manually.

# Repositories

Here are my repositories related to Spark:

* [github - hadoop\_spark](https://github.com/bulka4/hadoop_spark) – Running a multinode HDFS, Yarn and Spark cluster on Azure Linux VMs.

# Hadoop theory

## Materials

* [youtube - Data Engineering](https://www.youtube.com/watch?v=N6TmDNexxGI&list=PLGhXxbu7qYooyn_aWk1DqpIF1CjBzaSUn&index=2) – Hadoop theory
* [youtube - Data Engineering](https://www.youtube.com/watch?v=rsOSrEbK7sU&list=PLLa_h7BriLH1OE82WZOH534WufJq824mb) – The entire playlist about Hadoop
* [youtube - Data Engineering](https://www.youtube.com/watch?v=Tyg1FVNq40g&list=PLGhXxbu7qYooyn_aWk1DqpIF1CjBzaSUn&index=3) - Hadoop and Spark (9h video)

## Data blocks

Data blocks are a fixed size chunk of data into which files are split before being saved.

Each block is replicated to ensure fault tolerance (that we don’t loose data).

## Small files problem

HDFS has problems when we want to store in it a lot of small files. That’s because:

* Each data block can store only one file
  + For example if our file has 8mb and block size is 128mb, then the entire block will be used only for storing this one file, so the remaining 120mb will be unused.
* NameNode needs to handle metadata of all the files
  + There is only one NameNode which needs to handle metadata of all the files.
  + If there is too many files, there can be too much metadata for the NameNode to handle.
  + We can use the HDFS Federation to create multiple NameNodes.

## Client

Client is a software that users use to talk to HDFS. Clients are for example:

* HDFS CLI tool (the hdfs command)
* HDFS Java API
* Other hadoop tools (Spark, Hive, Pig)
* Third-party tools or GUIs that works with HDFS (Hue, Apache Nifi)

So users talk to a client and client talks to HDFS.

## Fsimage and edit logs

The fsimage file is a snapshot of the file system. It is also called the file system image.

The edit log file contains information about recent changes since the last snapshot.

Edit logs are merged into the fsimage periodically (checkpointing).

## Processes

In HDFS there is a few main processes (daemons, Java processes):

* NameNode
* DataNode
* Secondary NameNode

### NameNode

**Role**: Manages the file system namespace and metadata. It coordinates work between a client and DataNodes when reading / writing data.

**Responsibilities**:

* NameNode stores and provides information about:
  + directory structure, file names, permissions (metadata)
  + which DataNode has which data blocks
* Decides in which DataNodes to save data.

**Critical process**: If the NameNode is down, HDFS becomes inaccessible

### DataNode

**Role**: Stores actual data blocks on local disk.

**Responsibilities**:

* Sends heartbeats to NameNode to signal it's alive.
* Reads, writes, stores and replicates data as directed by the client and the NameNode

### Secondary NameNode

**Role**: Helps the primary NameNode handling edit logs. It does that by periodically merging edit logs with the file systemimage (fsimage).

**Important**:

* It does not replace the NameNode.
* Used to reduce load and prevent edit logs from growing too large

## Communication between users and processes

### User => client

Users always talk to client first and then client talks to other HDFS processes (NameNode, DataNode etc).

### Reading data

When client wants to read data, then communication looks like that:

* Client sends a request to the NameNode to read data.
* NameNode identifies which DataNodes have the data and sends that info back to the client.
* Client then contacts the proper DataNodes directly to read the data.

### Writing data

When a client wants to write data, then communication looks like this:

* The client sends a request to the NameNode to create a file.
* The NameNode selects appropriate DataNodes to store data (including replicas)
* The client pushes the data directly to the frist DataNode, which forwards it to other DataNodes to create the replicas.

No data is sent thorugh the NameNode. It only handles metadata and coordination.

## High Availability with Zookeeper

We can achieve a High Availability by having one Active NameNode and multiple Standby NameNodes.

Only the Active NameNode handles client requests and the other, Standby NameNodes are waiting to take over when the Active one fails.

Zookeeper can be used to monitor NameNodes and to replace a failed NameNode with another one.

## HDFS Federation

HDFS Federation allows to have multiple NameNodes, each managing part of the namespace.

Helps scale metadata storage horizontally.

Each NameNode operates independently, with its own set of DataNodes.

## Data Integrity

HDFS uses checksums to verify data integrity.

When data is written or read, HDFS validates it using these checksums.

Corrupt blocks are automatically detected and replaced from replicas.

### Checksums

A checksum is a short, fixed-size string of digits (often a hash) that uniquely represents the contents of a data block.

* It's generated using a hashing algorithm like CRC32, MD5, or SHA.

# Comparison to miniIO

## Separation of storage and compute

With miniIO it is easier to scale separately storage and compute. That’s because when you have data on one server and compute engine (like Spark) on another, then with miniIO it is not a problem.

Compute engine can get data stored on another server with miniIO efficiently over a network. On the other hand getting data from another server stored with HDFS is much less efficient because of networking design used for that.

## Speed of reading / writing data

When we have data stored in HDFS cluster, and we run for example Spark on the same cluster, then reading and writing data is faster than with miniIO.

But when Spark needs to read data from a server different then the one running Spark, then reading data from miniIO is faster then from HDFS.

It might be a good idea to keep all the historical data in the miniIO and in the HDFS cluster keep only data from the recent few years, which is currently used in analytics, and run Spark on the same cluster as HDFS for fast reads/writes.

# HDFS and Spark on Kubernetes

Running HDFS on Kubernetes is probably much more complicated than without it.

It might be a good idea to run Spark in a self managed Kubernetes cluster on a set of VMs and HDFS in the same cluster of VMs outside of Kubernetes.

# Debugging

## Hadoop logs

We can check logs in the hadoop/logs folder.

## Start different processes separately

When running the start-dfs.sh from the master node it starts multiple processes on different nodes.

If some processes doesn’t want to start we can try to start this one process on one specific node.

## SSH and networking

Hadoop uses TCP for communication between nodes. We can test it by running:

* Ping <hostname>

Also we need to test passwordless ssh communication from the master node to all the slave nodes.