# Pros

## Speed

HDFS is faster than compared to for example object storage. That’s because when you read / write data to object storage, that data goes over a network using HTTPS.

In case of HDFS data is accessed locally through TCP.

HDFS can be 2-3 times faster when reading / writing a lot of data compared to object storage.

## Fault tolerance

Built-in replication of data blocks ensures availability and durability.

# Cons

## Small files problem

It is not efficient for storing big number of small files. More info about that in another section ‘Hadoop theory > Small files problem’.

## Scaling

It is more difficult to scale (add more nodes) compared to object storage.

## Difficulty of setup

It is difficult to set up.

## Coupled storage and compute

HDFS was designed with tight coupling between storage and compute.

Scaling storage or compute independently is possible but requires manual tuning and breaks HDFS's natural behavior.

If we:

* **Try to access HDFS from outside the cluster → networking is difficult**

HDFS is not designed for external access. Exposing it over the network (e.g., to Spark on Kubernetes) requires configuring ports, clients, and security (especially with Kerberos).

* **Add compute- or storage-focused nodes → Resource mismatch risk**  
  Even if a node is compute-heavy but has little storage, HDFS will still store data on it. Likewise, a node with lots of storage but weak CPU/memory may still be assigned compute tasks (like Spark jobs), unless additional configuration is done. This makes HDFS harder to scale flexibly compared to object storage.

## Accessing data from outside of a cluster

If we want to access HDFS data from outside of a HDFS cluster, for example for reporting, it is possible but very complex (that’s related to decoupling storage and compute).