Lab 2 : HDFS File System Administration

Learning Objectives

- Start and validate a mini HDFS + YARN cluster under Docker.
- Practice HDFS administration tasks: users, permissions, ACLs, snapshots, quotas, replication, and integrity.
- Discover basic monitoring through the NameNode and ResourceManager web interfaces.

1) Launching the Cluster

```
docker compose up -d
docker compose ps
```

Verify that the following services are running: **namenode**, **datanode**, **resourcemanager**, and **nodemanager**.
If the NameNode starts for the first time, the variable `ENSURE_NAMENODE_DIR=/tmp/hadoop-root/dfs/name` forces initialization (format).

If there is an issue, check the logs:

```
docker logs -f namenode
docker logs -f datanode
```

2) Quick Checks via Web UI

- **NameNode UI (HDFS)**: http://localhost:9870
  → Check: number of live DataNodes, capacity, and block status.
- **YARN ResourceManager UI**: http://localhost:8088
  → Check: 1 NodeManager is “HEALTHY.”

Take screenshots as evidence.

3) Admin Shell in the NameNode

All the following HDFS commands are executed from inside the NameNode container:

```
docker exec -it namenode bash
# Then inside the container:
hdfs dfs -ls /
hdfs dfsadmin -report
```

4) User Tree and Permissions

Create a “cours” directory and simulate two logical users (without OS accounts):

```
hdfs dfs -mkdir -p /cours/tp1
hdfs dfs -mkdir -p /user/student1 /user/student2
hdfs dfs -chmod 755 /user /cours
hdfs dfs -chmod 700 /user/student1 /user/student2
```

Create a small sample dataset:

```
cat > /tmp/sales.csv << 'EOF'
id_client,date,total
101,2025-09-01,120.50
102,2025-09-01,45.00
101,2025-09-02,33.20
103,2025-09-02,250.00
EOF

hdfs dfs -put -f /tmp/sales.csv /cours/tp1/
hdfs dfs -ls -h /cours/tp1
```

5) ACLs (Fine-Grained Access Control) on HDFS

Objective: give **student2** read-only access to `/cours/tp1` without changing standard permissions.

```
# Display existing ACLs
hdfs dfs -getfacl /cours/tp1

# Add read (r-x) ACL for "student2"
hdfs dfs -setfacl -m user:student2:r-x /cours/tp1

# Verify
hdfs dfs -getfacl /cours/tp1
```

If “student2” doesn’t exist on the host system, HDFS still stores it symbolically.
For demonstration, focus on how ACLs work and the resulting access behavior.

6) Snapshots (Logical Directory Backup)

Enable snapshots on `/cours` and create one before modification:

```
hdfs dfsadmin -allowSnapshot /cours
SNAP="pre_modif_$(date +%Y%m%d_%H%M%S)"
hdfs dfs -createSnapshot /cours "$SNAP"
hdfs dfs -ls /cours/.snapshot
```

Test accidental deletion and restoration:

```
hdfs dfs -rm /cours/tp1/sales.csv
hdfs dfs -ls /cours/tp1

# Restore from snapshot
hdfs dfs -cp /cours/.snapshot/$SNAP/tp1/sales.csv /cours/tp1/
hdfs dfs -ls /cours/tp1
```

7) Quotas (Space & File Limits)

Set a quota on `/user/student1`:

```
hdfs dfsadmin -setSpaceQuota 1m /user/student1
hdfs dfsadmin -setQuota 100 /user/student1
hdfs dfs -count -q -h /user/student1
```

Test by copying a slightly large file:

```
dd if=/dev/zero of=/tmp/bigfile.bin bs=64K count=40  # ~2.5 MB
hdfs dfs -put /tmp/bigfile.bin /user/student1/ || echo "Expected failure
(quota exceeded)"
```

Reset if needed:

```
hdfs dfsadmin -clrSpaceQuota /user/student1
hdfs dfsadmin -clrQuota /user/student1
```

8) Replication Factor & Data Integrity

Check replication settings:

```
# Default replication factor
hdfs getconf -confKey dfs.replication || true

# For a specific file
hdfs dfs -stat %r /cours/tp1/sales.csv
```

Change the replication factor (e.g., from 1 → 2) and optionally run the balancer:

```
hdfs dfs -setrep 2 /cours/tp1/sales.csv
hdfs dfs -stat %r /cours/tp1/sales.csv
hdfs balancer -threshold 10
```

Check integrity:

```
hdfs fsck / -files -blocks -locations | head -n 50
```

9) Safemode and Admin Operations

Show the “safemode” (read-only NameNode state):

```
hdfs dfsadmin -safemode get
```

10) Cluster Monitoring with Prometheus and Grafana

Objective

Deploy a monitoring system for Hadoop components (NameNode & DataNodes) using **Prometheus** and **Grafana**, to visualize HDFS metrics: capacity, blocks, files, and JVM health.

10.1 Monitoring Architecture

- **JMX Exporter** on each Hadoop service (NameNode, DataNode) exposes JVM and HDFS metrics.
- **Prometheus** collects the metrics.
- **Grafana** visualizes them as dashboards and charts.

```
NameNode (JMX) ─┐
                ├──▶ Prometheus ───▶ Grafana
DataNode (JMX) ─┘
```

10.2 JMX Configuration for HDFS

(File: `monitoring/jmx/hadoop.yml`)
Contains metric mapping rules for HDFS and JVM.

10.3 Prometheus & Grafana Integration

Add to `docker-compose.yml`:

```
prometheus:
  image: prom/prometheus
  container_name: prometheus
  ports:
    - "9090:9090"
  volumes:
    - ./monitoring/prometheus.yml:/etc/prometheus/prometheus.yml
  networks:
    - hadoop

grafana:
  image: grafana/grafana
  container_name: grafana
  ports:
    - "3000:3000"
  environment:
    - GF_SECURITY_ADMIN_USER=admin
    - GF_SECURITY_ADMIN_PASSWORD=admin
  depends_on:
    - prometheus
  networks:
    - hadoop
```

Prometheus configuration (`monitoring/prometheus.yml`) defines scraping jobs for NameNode and DataNode.

Launch monitoring:

```
docker compose up -d
```

Access:

- Prometheus → http://localhost:9090
- Grafana → http://localhost:3000 (admin/admin)

10.4 Grafana Dashboard

Import the `hdfs-mini-dashboard.json` file and visualize:

- HDFS Capacity Used
- HDFS Capacity Remaining
- HDFS Blocks Total
- HDFS Files Total
- JVM Heap Used
- JVM Threads Current

10.5 Experimentation

```
hdfs dfs -mkdir /test
hdfs dfs -put /etc/passwd /test/
hdfs dfs -put /etc/hosts /test/
```

Then refresh Grafana to see metrics evolve.

11) Prometheus Alert System Integration

Objective

Set up automatic alerts for real-time cluster issues: DataNode down, NameNode in safemode, or low disk space.

Add **Alertmanager** to your Docker Compose, configure Prometheus rules and alert routes, and visualize them in Grafana.

Simulate a failure:

```
docker stop datanode
```

→ After 30 seconds, the **DataNodeDown** alert appears.

Restart:

```
docker start datanode
```

→ The alert automatically resolves.