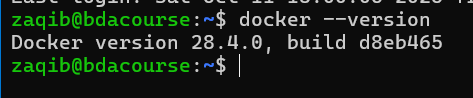
# Hadoop Word Count Lab

#### Note: The commands are all in red. The tasks required are highlighted in yellow. You should also explain the snapshots required.

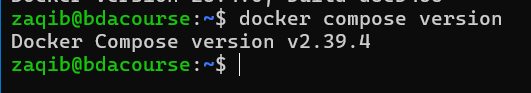
### Prerequisites Check

# Check Docker is installed and running

docker –version



docker compose version



# Check if you have docker compose (v2) or docker-compose (v1)

# Use 'docker compose' for v2 or 'docker-compose' for v1

docker compose version

### STEP 1: Complete Cleanup (Run this first!)

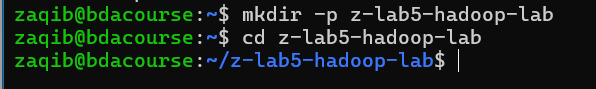
**What this does:** Stops all Hadoop containers, removes old data directories, and cleans up previous lab attempts to ensure a fresh start.

# Navigate to your working directory (adjust path as needed)

cd ~

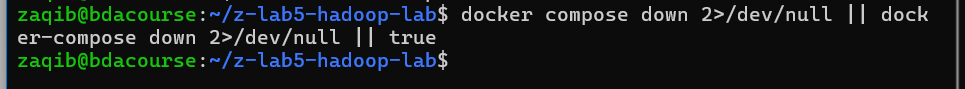
mkdir -p hadoop-lab

cd hadoop-lab



# Stop and remove any running Hadoop containers

docker compose down 2>/dev/null || docker-compose down 2>/dev/null || true



# Remove all Hadoop-related containers forcefully

docker rm -f namenode datanode1 datanode2 datanode3 resourcemanager nodemanager1 nodemanager2 nodemanager3 historyserver 2>/dev/null || true



# Remove old data directories

sudo rm -rf ./data

# Remove old compose file

rm -f docker-compose.yml docker-compose.phase2.yml

# Remove old result files

rm -f part-r-00000\_\*

echo "Cleanup complete!"



### STEP 2: Create Docker Compose File (if indentation issues occur, seek help from gpt)

**What this does:** Creates a small Hadoop cluster (HDFS + YARN + MapReduce history server) using the bde2020 Docker images

* NameNode (master for HDFS metadata + web UI)
* 3 DataNodes (store HDFS blocks)
* ResourceManager (YARN central scheduler + web UI)
* 3 NodeManagers (YARN workers, run containers; provide shuffle service)
* HistoryServer (MapReduce job history UI)
* All containers are attached to a single Docker bridge network hadoop-net

version: "3.8"

services:

namenode:

image: bde2020/hadoop-namenode:2.0.0-hadoop3.2.1-java8

container\_name: namenode

hostname: namenode

ports:

- "9870:9870"

- "9000:9000"

- "8020:8020"

volumes:

- ./data/namenode:/hadoop/dfs/name

environment:

- CLUSTER\_NAME=hadoop-cluster

- CORE\_CONF\_fs\_defaultFS=hdfs://namenode:8020

- CORE\_CONF\_hadoop\_http\_staticuser\_user=root

- HDFS\_CONF\_dfs\_namenode\_rpc\_bind\_host=0.0.0.0

- HDFS\_CONF\_dfs\_namenode\_servicerpc\_bind\_host=0.0.0.0

- HDFS\_CONF\_dfs\_namenode\_http\_bind\_host=0.0.0.0

- HDFS\_CONF\_dfs\_namenode\_https\_bind\_host=0.0.0.0

- HDFS\_CONF\_dfs\_replication=3

- HDFS\_CONF\_dfs\_namenode\_datanode\_registration\_ip\_\_\_hostname\_\_\_check=false

- YARN\_CONF\_yarn\_resourcemanager\_address=resourcemanager:8035

- YARN\_CONF\_yarn\_resourcemanager\_hostname=resourcemanager

networks:

- hadoop-net

healthcheck:

test: ["CMD", "curl", "-f", "http://localhost:9870"]

interval: 30s

timeout: 10s

retries: 5

datanode1:

image: bde2020/hadoop-datanode:2.0.0-hadoop3.2.1-java8

container\_name: datanode1

hostname: datanode1

depends\_on:

- namenode

ports:

- "9864:9864"

volumes:

- ./data/datanode1:/hadoop/dfs/data

environment:

- CLUSTER\_NAME=hadoop-cluster

- CORE\_CONF\_fs\_defaultFS=hdfs://namenode:8020

- HDFS\_CONF\_dfs\_datanode\_address=0.0.0.0:9866

- HDFS\_CONF\_dfs\_datanode\_http\_address=0.0.0.0:9864

- HDFS\_CONF\_dfs\_datanode\_ipc\_address=0.0.0.0:9867

networks:

- hadoop-net

restart: unless-stopped

datanode2:

image: bde2020/hadoop-datanode:2.0.0-hadoop3.2.1-java8

container\_name: datanode2

hostname: datanode2

depends\_on:

- namenode

ports:

- "9865:9864"

volumes:

- ./data/datanode2:/hadoop/dfs/data

environment:

- CLUSTER\_NAME=hadoop-cluster

- CORE\_CONF\_fs\_defaultFS=hdfs://namenode:8020

- HDFS\_CONF\_dfs\_datanode\_address=0.0.0.0:9866

- HDFS\_CONF\_dfs\_datanode\_http\_address=0.0.0.0:9864

- HDFS\_CONF\_dfs\_datanode\_ipc\_address=0.0.0.0:9867

networks:

- hadoop-net

restart: unless-stopped

datanode3:

image: bde2020/hadoop-datanode:2.0.0-hadoop3.2.1-java8

container\_name: datanode3

hostname: datanode3

depends\_on:

- namenode

ports:

- "9866:9864"

volumes:

- ./data/datanode3:/hadoop/dfs/data

environment:

- CLUSTER\_NAME=hadoop-cluster

- CORE\_CONF\_fs\_defaultFS=hdfs://namenode:8020

- HDFS\_CONF\_dfs\_datanode\_address=0.0.0.0:9866

- HDFS\_CONF\_dfs\_datanode\_http\_address=0.0.0.0:9864

- HDFS\_CONF\_dfs\_datanode\_ipc\_address=0.0.0.0:9867

networks:

- hadoop-net

restart: unless-stopped

resourcemanager:

image: bde2020/hadoop-resourcemanager:2.0.0-hadoop3.2.1-java8

container\_name: resourcemanager

hostname: resourcemanager

ports:

- "8088:8088"

- "8035:8035" # <-- expose new RM RPC port on host

depends\_on:

- namenode

environment:

- CLUSTER\_NAME=hadoop-cluster

- CORE\_CONF\_fs\_defaultFS=hdfs://namenode:8020

- YARN\_CONF\_yarn\_resourcemanager\_hostname=resourcemanager

- YARN\_CONF\_yarn\_resourcemanager\_address=resourcemanager:8035 # <-- new address

networks:

- hadoop-net

restart: unless-stopped

nodemanager1:

image: bde2020/hadoop-nodemanager:2.0.0-hadoop3.2.1-java8

container\_name: nodemanager1

hostname: nodemanager1

depends\_on:

- resourcemanager

ports:

- "8042:8042"

environment:

- CLUSTER\_NAME=hadoop-cluster

- CORE\_CONF\_fs\_defaultFS=hdfs://namenode:8020

- YARN\_CONF\_yarn\_resourcemanager\_hostname=resourcemanager

- YARN\_CONF\_yarn\_resourcemanager\_address=resourcemanager:8035

- YARN\_CONF\_yarn\_nodemanager\_aux\_\_\_services=mapreduce\_shuffle

- YARN\_CONF\_yarn\_nodemanager\_resource\_memory\_\_\_mb=4096

- YARN\_CONF\_yarn\_nodemanager\_resource\_cpu\_\_\_vcores=2

networks:

- hadoop-net

restart: unless-stopped

nodemanager2:

image: bde2020/hadoop-nodemanager:2.0.0-hadoop3.2.1-java8

container\_name: nodemanager2

hostname: nodemanager2

depends\_on:

- resourcemanager

ports:

- "8043:8042"

environment:

- CLUSTER\_NAME=hadoop-cluster

- CORE\_CONF\_fs\_defaultFS=hdfs://namenode:8020

- YARN\_CONF\_yarn\_resourcemanager\_hostname=resourcemanager

- YARN\_CONF\_yarn\_resourcemanager\_address=resourcemanager:8035

- YARN\_CONF\_yarn\_nodemanager\_aux\_\_\_services=mapreduce\_shuffle

- YARN\_CONF\_yarn\_nodemanager\_resource\_memory\_\_\_mb=4096

- YARN\_CONF\_yarn\_nodemanager\_resource\_cpu\_\_\_vcores=2

networks:

- hadoop-net

restart: unless-stopped

nodemanager3:

image: bde2020/hadoop-nodemanager:2.0.0-hadoop3.2.1-java8

container\_name: nodemanager3

hostname: nodemanager3

depends\_on:

- resourcemanager

ports:

- "8044:8042"

environment:

- CLUSTER\_NAME=hadoop-cluster

- CORE\_CONF\_fs\_defaultFS=hdfs://namenode:8020

- YARN\_CONF\_yarn\_resourcemanager\_hostname=resourcemanager

- YARN\_CONF\_yarn\_resourcemanager\_address=resourcemanager:8035

- YARN\_CONF\_yarn\_nodemanager\_aux\_\_\_services=mapreduce\_shuffle

- YARN\_CONF\_yarn\_nodemanager\_resource\_memory\_\_\_mb=4096

- YARN\_CONF\_yarn\_nodemanager\_resource\_cpu\_\_\_vcores=2

networks:

- hadoop-net

restart: unless-stopped

historyserver:

image: bde2020/hadoop-historyserver:2.0.0-hadoop3.2.1-java8

container\_name: historyserver

hostname: historyserver

ports:

- "8188:8188"

depends\_on:

- namenode

- resourcemanager

environment:

- CLUSTER\_NAME=hadoop-cluster

- CORE\_CONF\_fs\_defaultFS=hdfs://namenode:8020

- YARN\_CONF\_yarn\_resourcemanager\_address=resourcemanager:8035

- YARN\_CONF\_yarn\_resourcemanager\_hostname=resourcemanager

networks:

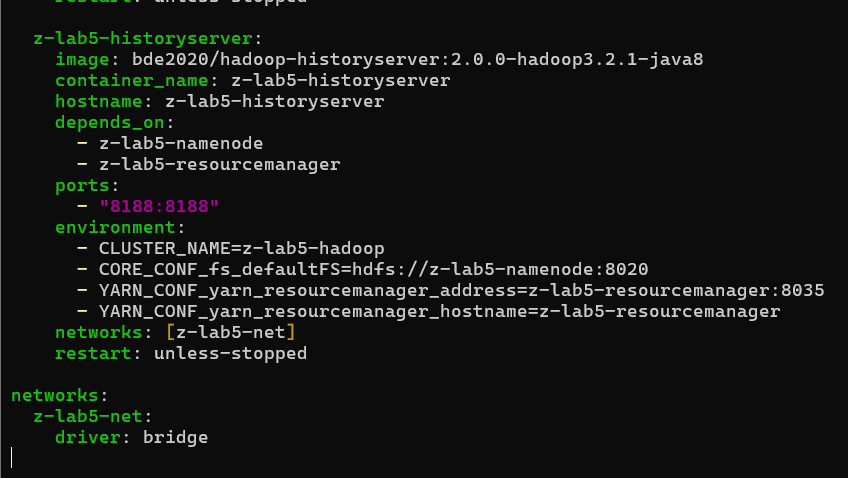
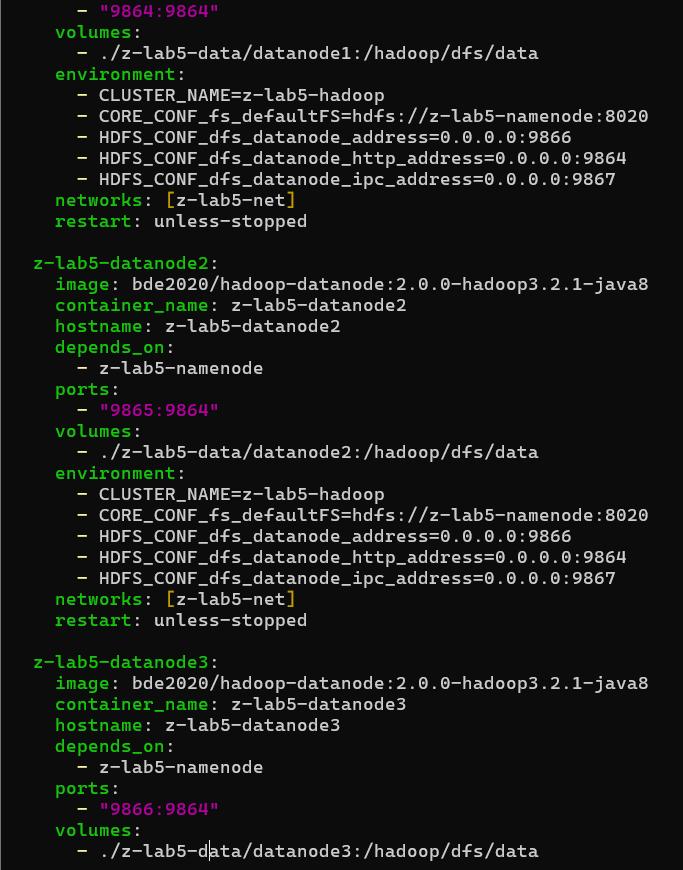
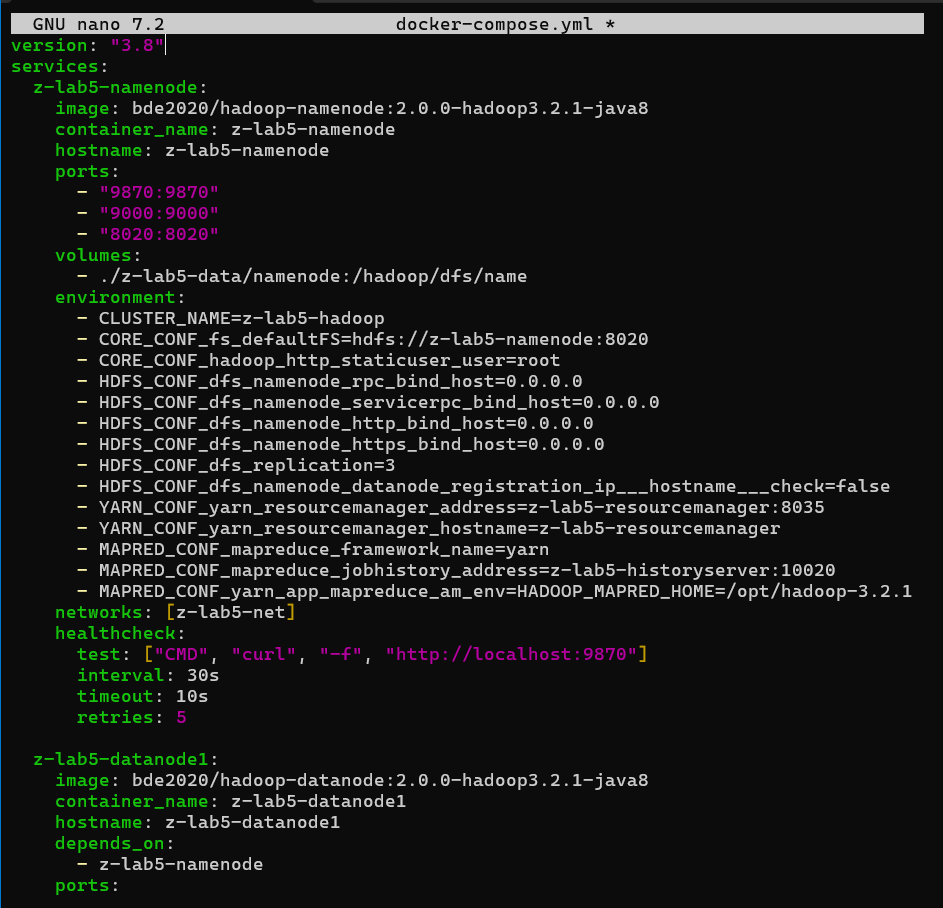
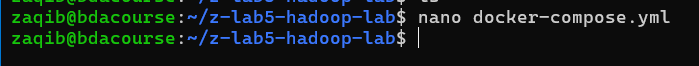
- hadoop-net

restart: unless-stopped

networks:

hadoop-net:

driver: bridge



To make sure the application log appears and it doesnt run locally, but runs on yarn:

- MAPRED\_CONF\_mapreduce\_framework\_name=yarn

- MAPRED\_CONF\_mapreduce\_jobhistory\_address=historyserver:10020

- MAPRED\_CONF\_yarn\_app\_mapreduce\_am\_env=HADOOP\_MAPRED\_HOME=/opt/hadoop-3.2.1

### STEP 3: Create Base Words File

**What this does:** Creates a base text file with technical words that will be used as input for WordCount. You can replace this with your own words.txt if needed.

# Create a sample words.txt file with diverse content

cat > words.txt << 'EOF'

hadoop mapreduce distributed computing framework apache big data processing cluster parallel execution hdfs namenode datanode replication block storage yarn resourcemanager nodemanager container scheduling java programming language platform development linux ubuntu terminal command shell script docker container virtualization orchestration compose

network communication protocol tcp ip address database sql nosql query transaction acid

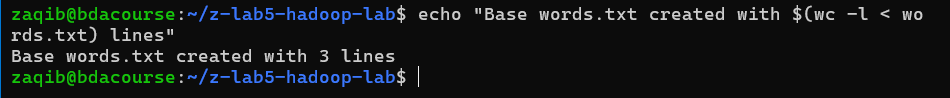
cloud computing infrastructure service platform machine learning algorithm model training prediction artificial intelligence neural network deep learning software engineering design pattern architecture web application frontend backend api rest security encryption authentication authorization performance optimization scalability throughput latency

EOF

A screenshot of a computer program

AI-generated content may be incorrect.

echo "Base words.txt created with $(wc -l < words.txt) lines"

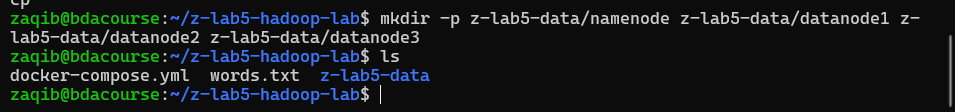


### STEP 4: Start Hadoop Cluster

**What this does:** Creates necessary directories, starts all Hadoop services in detached mode, waits for initialization, and verifies all containers are running.

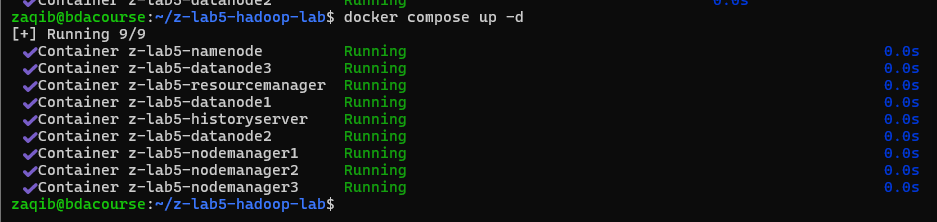
# Create data directories

mkdir -p data/namenode data/datanode1 data/datanode2 data/datanode3



# Start all containers

docker compose up -d



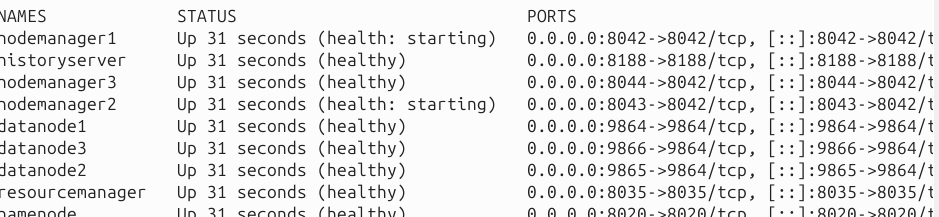
# Wait for services to initialize (30 seconds)

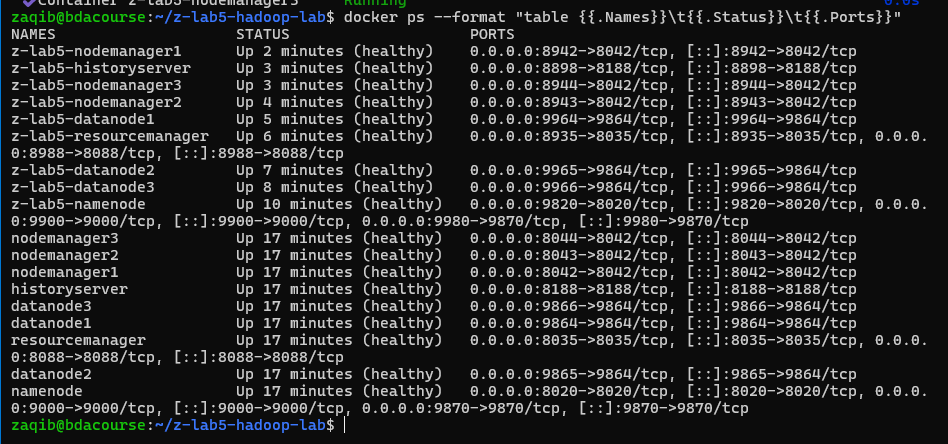
echo "Waiting for Hadoop services to start..."

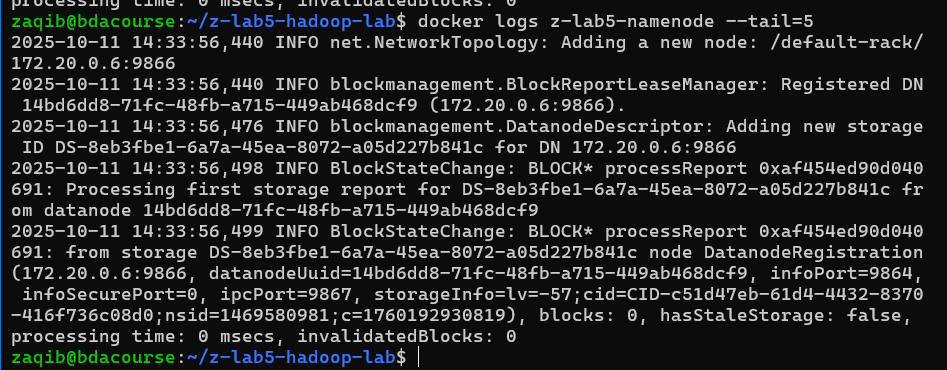
sleep 30

# Check all containers are running

docker ps --format "table {{.Names}}\t{{.Status}}\t{{.Ports}}"





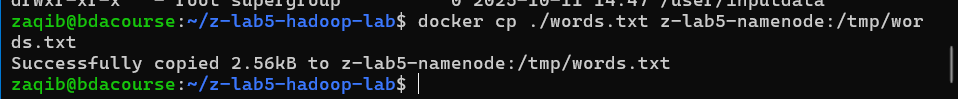


### STEP 5: Copy Words File to NameNode

**What this does:** Copies your base words.txt file from the Ubuntu host into the namenode container's /tmp directory.

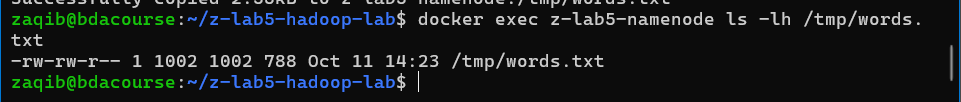
# Copy words.txt into the namenode container

docker cp words.txt namenode:/tmp/words.txt

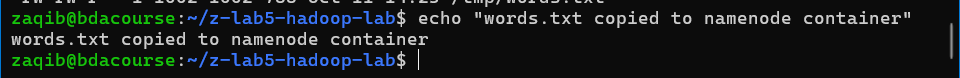


# Verify the file was copied

docker exec namenode ls -lh /tmp/words.txt



echo "words.txt copied to namenode container"



### STEP 6: Generate Large Dataset Files Inside NameNode

**What this does:** Creates four datasets inside the namenode container:

* **set0**: Original small file
* **set1**: 1 GB file (by repeatedly appending base file)
* **set2**: 1.5 GB file
* **set3**: 2 GB file

**Note:** This may take 5-15 minutes depending on your system. The script repeatedly concatenates the base file until reaching target sizes. You can ask gpt for a faster method.

# Execute all commands inside the namenode container

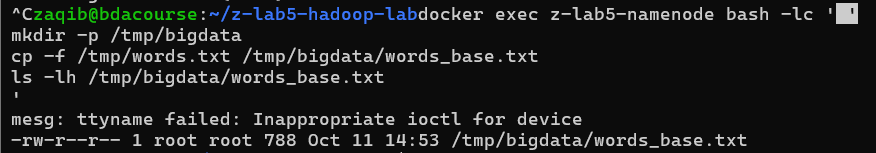
docker exec namenode bash -c '

# Create bigdata directory

mkdir -p /tmp/bigdata

# Copy base file

cp /tmp/words.txt /tmp/bigdata/words\_base.txt



cd /tmp/bigdata

# Create set0 (original small file)

cp words\_base.txt words\_set0.txt

# Create set1 (1 GB)

echo "Creating 1 GB file (set1)..."

rm -f words\_set1.txt

cp words\_base.txt words\_set1.txt

target1=$((1024\*1024\*1024))

while [ $(stat -c%s words\_set1.txt) -lt $target1 ]; do

cat words\_base.txt >> words\_set1.txt

done

# Create set2 (1.5 GB)

echo "Creating 1.5 GB file (set2)..."

rm -f words\_set2.txt

cp words\_base.txt words\_set2.txt

target2=$((1536\*1024\*1024))

while [ $(stat -c%s words\_set2.txt) -lt $target2 ]; do

cat words\_base.txt >> words\_set2.txt

done

# Create set3 (2 GB)

echo "Creating 2 GB file (set3)..."

rm -f words\_set3.txt

cp words\_base.txt words\_set3.txt

target3=$((2048\*1024\*1024))

while [ $(stat -c%s words\_set3.txt) -lt $target3 ]; do

cat words\_base.txt >> words\_set3.txt

done

# Display file sizes

echo ""

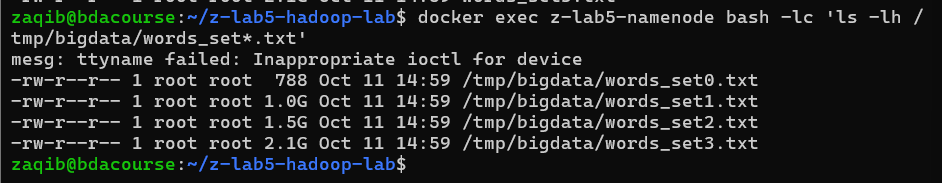
echo "Generated files:"

ls -lh words\_set\*.txt

'

A screen shot of a computer program

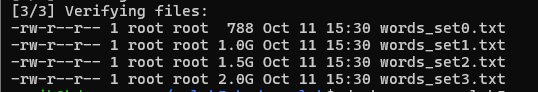
AI-generated content may be incorrect.

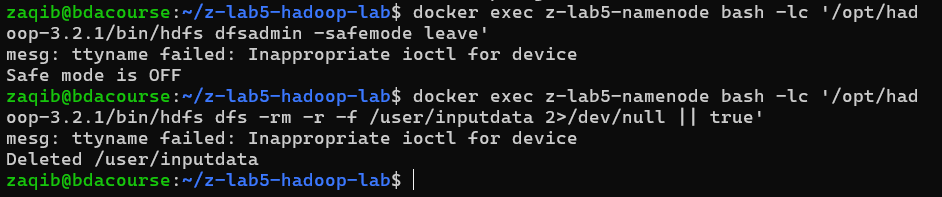


### STEP 7: Upload Files to HDFS

**What this does:**

* Cleans up any old HDFS directories from previous lab runs
* Creates separate HDFS directories for each dataset
* Uploads all four dataset files to HDFS
* Displays the final HDFS directory structure





docker exec namenode bash -c '

# Remove any existing HDFS directories from previous runs

hdfs dfs -rm -r -f /user/inputdata 2>/dev/null || true

hdfs dfs -rm -r -f /user/output\_wc\_set0 2>/dev/null || true

hdfs dfs -rm -r -f /user/output\_wc\_set1 2>/dev/null || true

hdfs dfs -rm -r -f /user/output\_wc\_set2 2>/dev/null || true

hdfs dfs -rm -r -f /user/output\_wc\_set3 2>/dev/null || true

A screen shot of a computer program

AI-generated content may be incorrect.

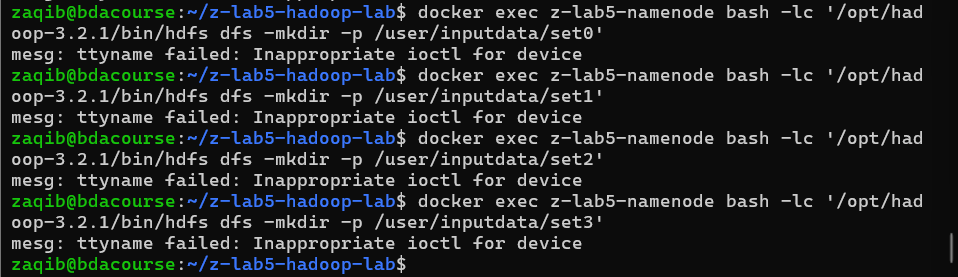
# Create HDFS directories for each dataset

hdfs dfs -mkdir -p /user/inputdata/set0

hdfs dfs -mkdir -p /user/inputdata/set1

hdfs dfs -mkdir -p /user/inputdata/set2

hdfs dfs -mkdir -p /user/inputdata/set3

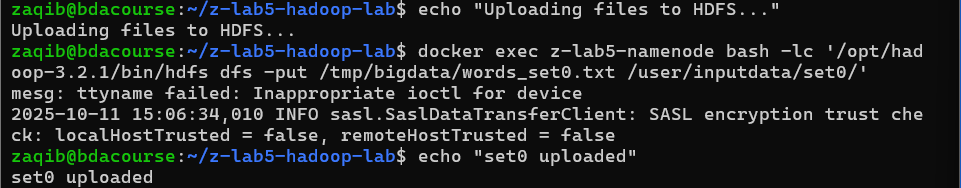


echo "Uploading files to HDFS..."

# Upload files to HDFS (this may take time for large files)

hdfs dfs -put /tmp/bigdata/words\_set0.txt /user/inputdata/set0/

echo "set0 uploaded"



hdfs dfs -put /tmp/bigdata/words\_set1.txt /user/inputdata/set1/

echo "set1 uploaded"

A screen shot of a computer

AI-generated content may be incorrect.

hdfs dfs -put /tmp/bigdata/words\_set2.txt /user/inputdata/set2/

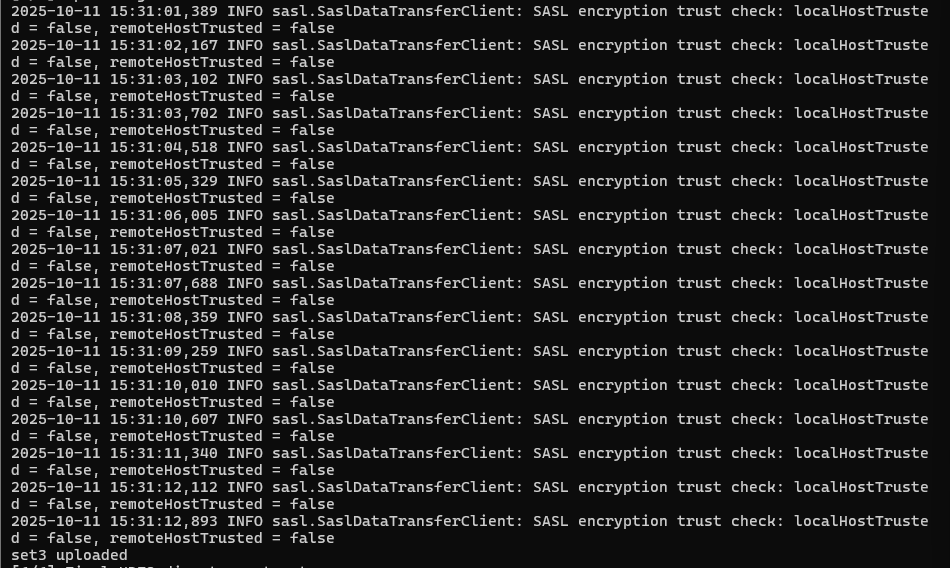
echo "set2 uploaded"

A screen shot of a computer

AI-generated content may be incorrect.

hdfs dfs -put /tmp/bigdata/words\_set3.txt /user/inputdata/set3/

echo "set3 uploaded"

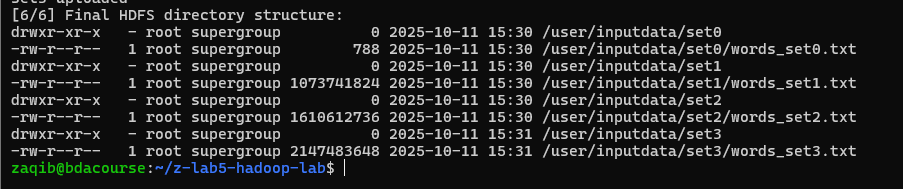


echo ""

echo "HDFS directory structure:"

hdfs dfs -ls -R /user/inputdata

'



**SCREENSHOT 1:** Capture the output showing the HDFS directory listing with all uploaded files.

### STEP 8: Locate Hadoop Examples JAR

**What this does:** Searches for the Hadoop MapReduce examples JAR file (which contains the WordCount program) and copies it to /tmp for easier access.

docker exec namenode bash -c '

echo "Searching for Hadoop MapReduce examples JAR..."

find /opt/hadoop\* /usr/local/hadoop\* -name "\*examples\*.jar" 2>/dev/null | head -5

# Copy the examples jar to /tmp for easy access

JAR\_PATH=$(find /opt/hadoop\* /usr/local/hadoop\* -name "\*examples\*.jar" 2>/dev/null | grep -v sources | head -1)

if [ -n "$JAR\_PATH" ]; then

cp "$JAR\_PATH" /tmp/hadoop-examples.jar

echo ""

echo "Examples JAR copied to /tmp/hadoop-examples.jar"

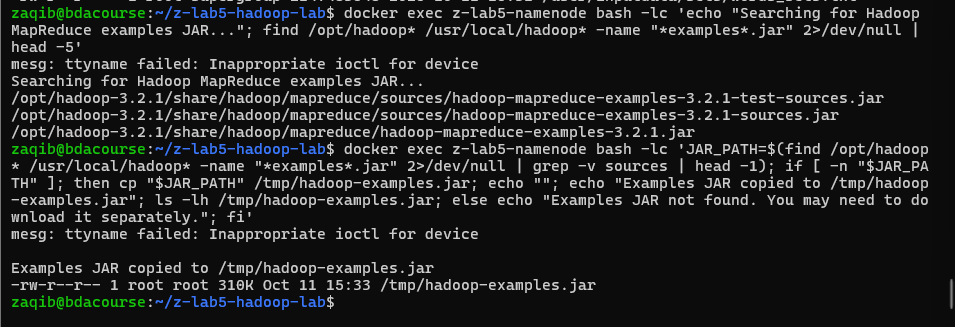
ls -lh /tmp/hadoop-examples.jar

else

echo "Examples JAR not found. You may need to download it separately."

fi

'



### STEP 9: Run WordCount on All Datasets

**What this does:**

* Runs the WordCount MapReduce job on all four datasets sequentially
* Measures and displays execution time for each job
* Removes previous output directories before each run
* Copies the final word count results to /tmp inside the container

docker exec namenode bash -c '

JAR=/tmp/hadoop-examples.jar

OUTPUT\_BASE=/user/output\_wc

# Function to run WordCount and measure time

run\_wordcount() {

dataset=$1

echo "━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━"

echo "Running WordCount on $dataset"

echo "━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━"

# Remove old output

hdfs dfs -rm -r -f ${OUTPUT\_BASE}\_${dataset} 2>/dev/null || true

# Record start time

START=$(date +%s)

# Run WordCount

hadoop jar $JAR wordcount /user/inputdata/${dataset} ${OUTPUT\_BASE}\_${dataset}

# Record end time

END=$(date +%s)

DURATION=$((END - START))

echo ""

echo "$dataset completed in $DURATION seconds"

echo ""

# Copy output to local tmp

hdfs dfs -get ${OUTPUT\_BASE}\_${dataset}/part-r-00000 /tmp/part-r-00000\_${dataset} 2>/dev/null || true

}

# Run WordCount on all datasets

run\_wordcount "set0"

run\_wordcount "set1"

run\_wordcount "set2"

run\_wordcount "set3"

echo "━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━"

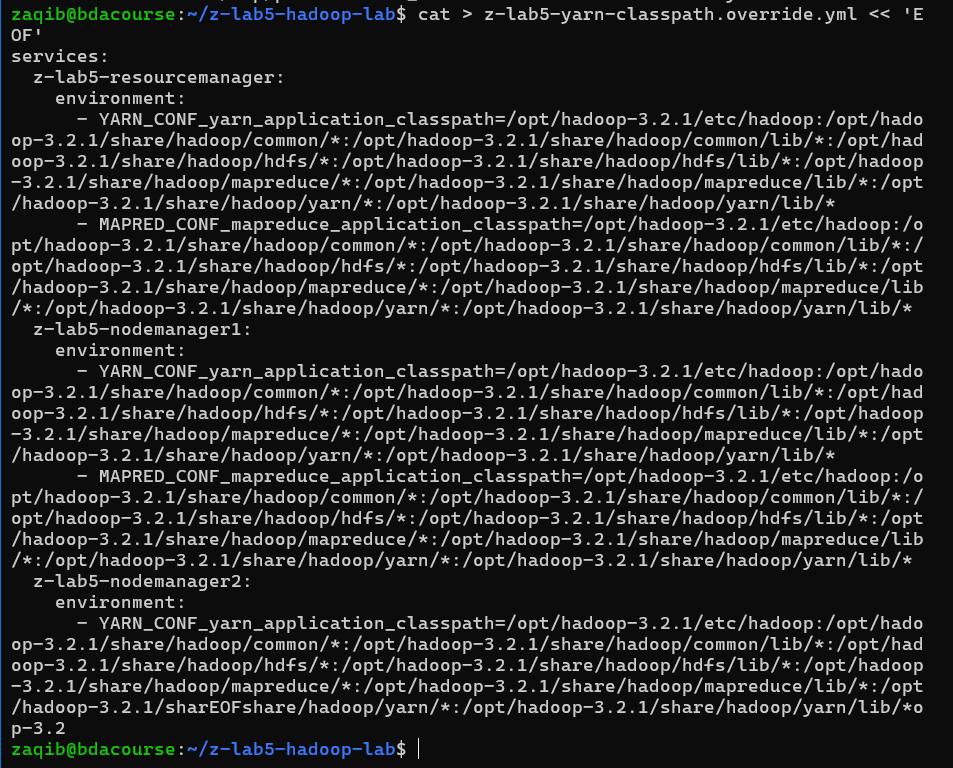
echo "All WordCount jobs completed!"

echo "━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━"

# Show output files

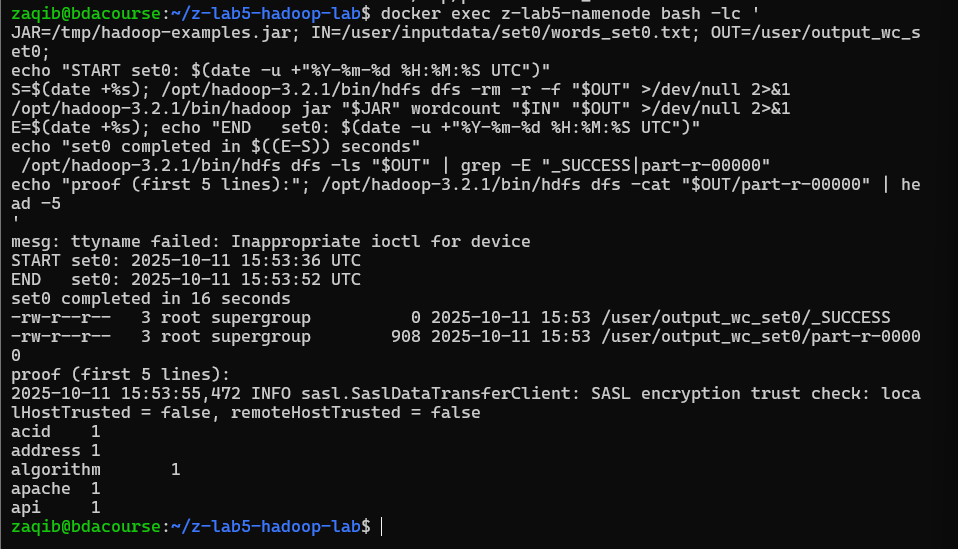
ls -lh /tmp/part-r-00000\_\*

'

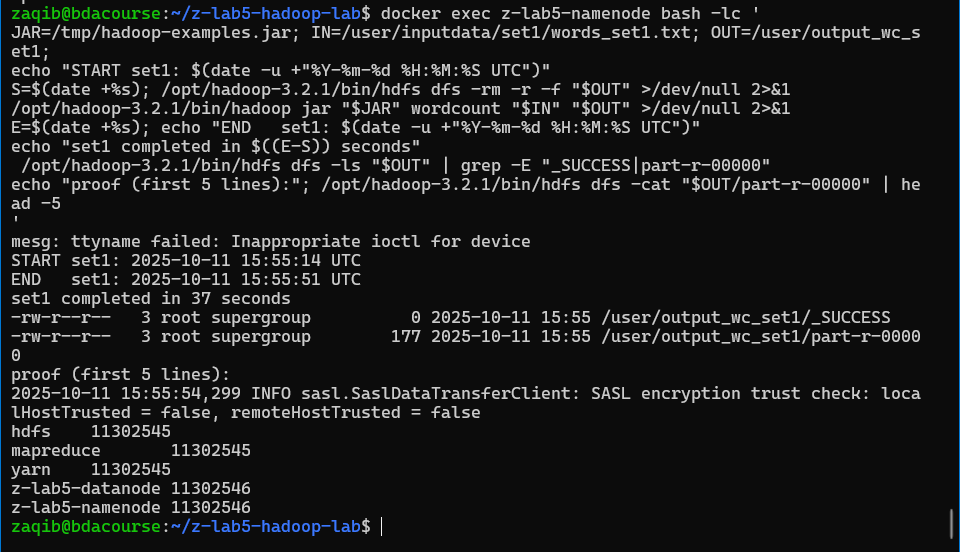


**SCREENSHOT 2-5:** Capture the completion message and execution time for each dataset (set0, set1, set2, set3).

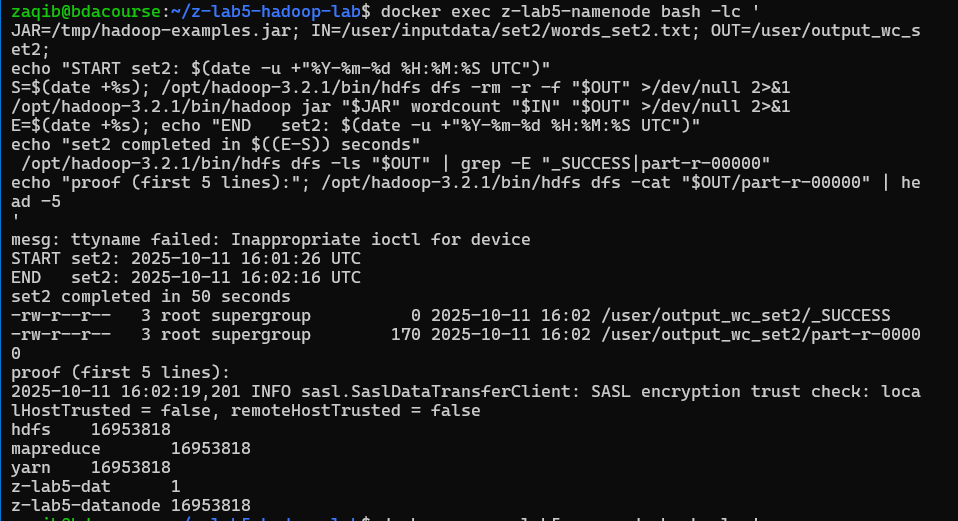
****





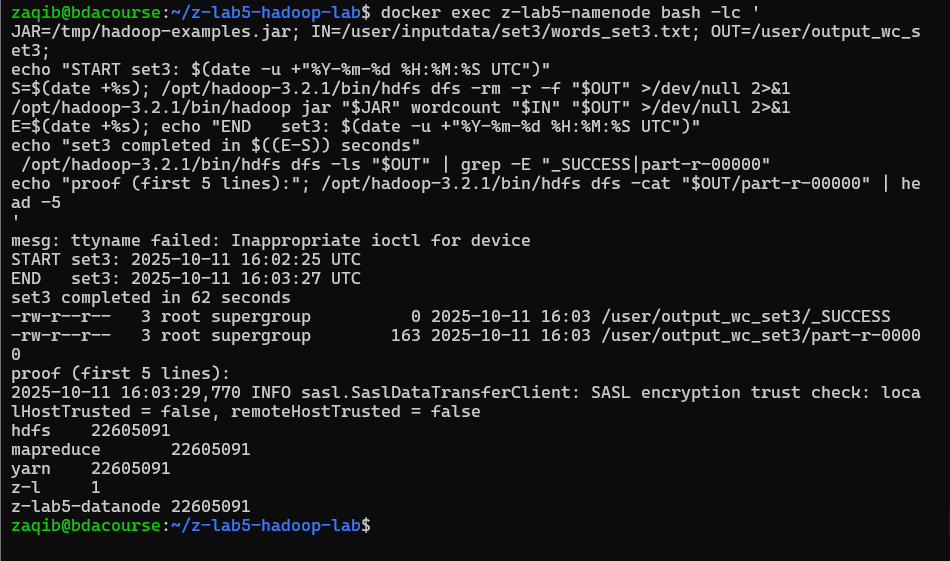


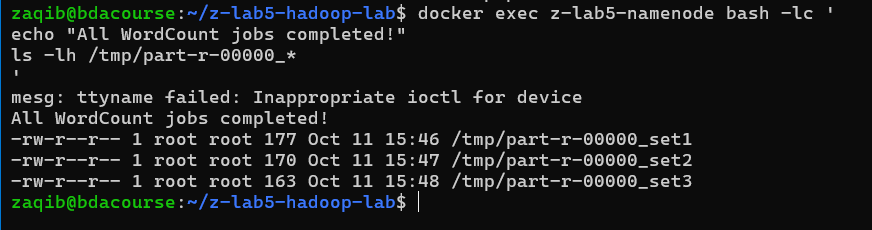




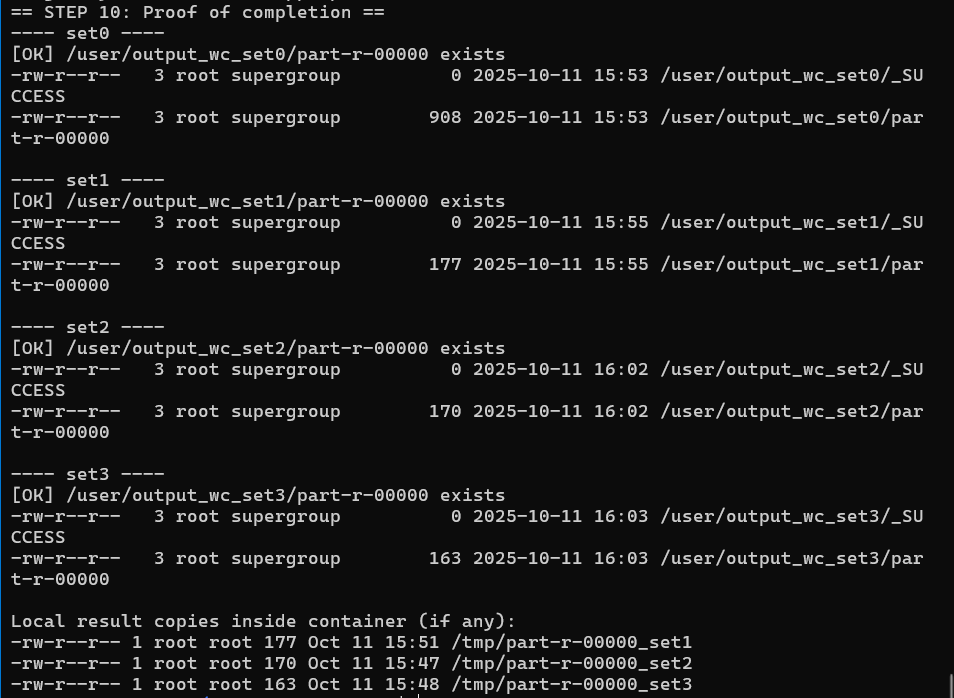
A black background with white text

AI-generated content may be incorrect.



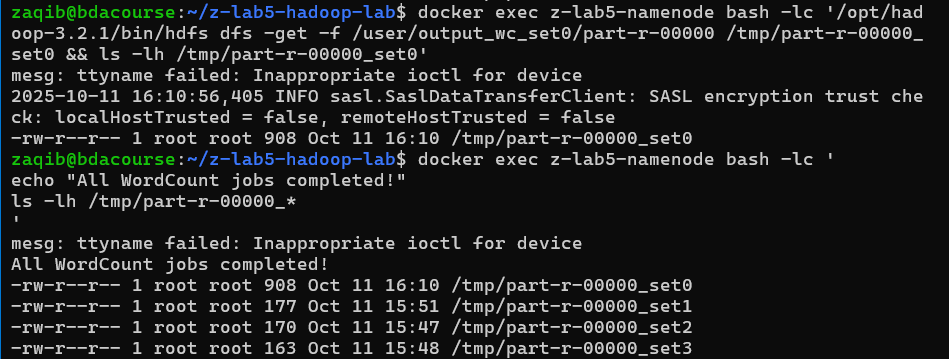


* **Expected times:** set0 (~30 sec), set1 (~5-10 min), set2 (~10-15 min), set3 (~15-20 min) depending on your system.

****

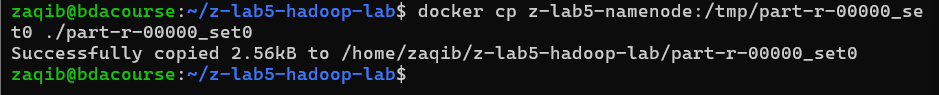
### STEP 10: Copy Results to Your Ubuntu Host

**What this does:** Copies all WordCount result files from the namenode container to your Ubuntu host machine, then displays a preview of the results.

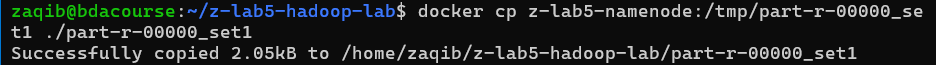


# Copy all result files from namenode to current directory

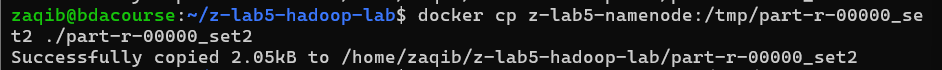
docker cp namenode:/tmp/part-r-00000\_set0 ./part-r-00000\_set0



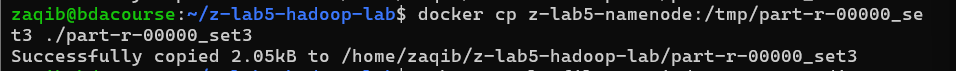
docker cp namenode:/tmp/part-r-00000\_set1 ./part-r-00000\_set1



docker cp namenode:/tmp/part-r-00000\_set2 ./part-r-00000\_set2



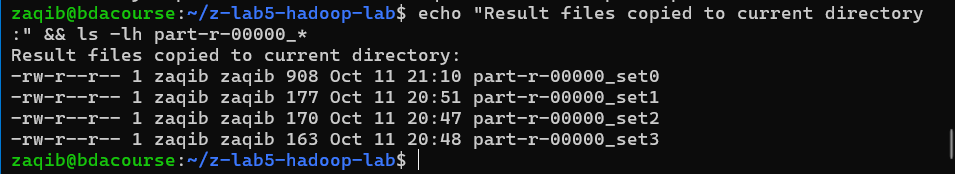
docker cp namenode:/tmp/part-r-00000\_set3 ./part-r-00000\_set3



# Display the results

echo "Result files copied to current directory:"

ls -lh part-r-00000\_\*



# Preview the word counts from largest dataset

echo ""

echo "Sample output from set3 (first 20 lines):"

head -20 part-r-00000\_set3

A computer screen shot of a black screen

AI-generated content may be incorrect.

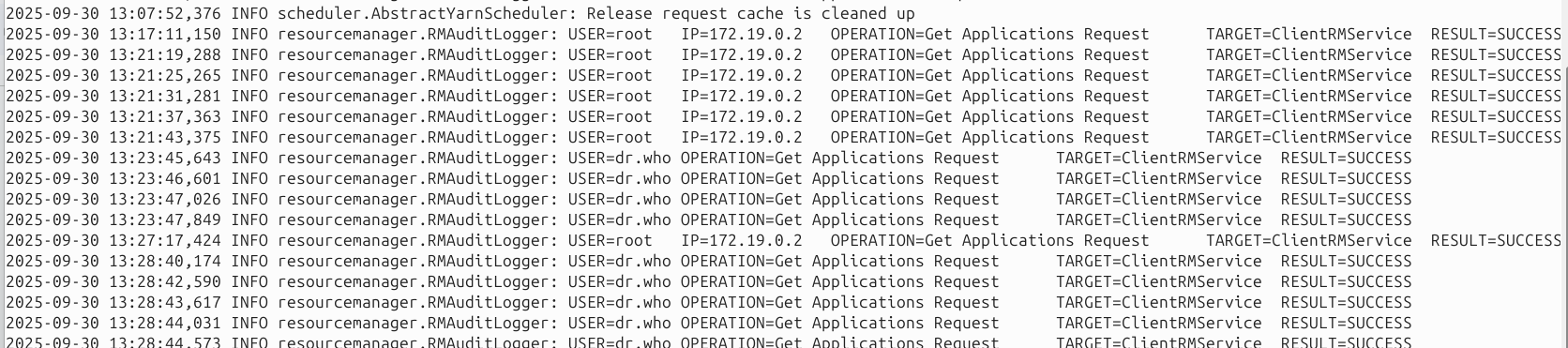
WHY IT IS ONLY 8 LINES BECAUSE THE WORDS WE GENERATED WAS ONLY 8 TOKENS AND IT REPEATS. CHECK THIS PREVIOUS OUTPUT PIC HERE

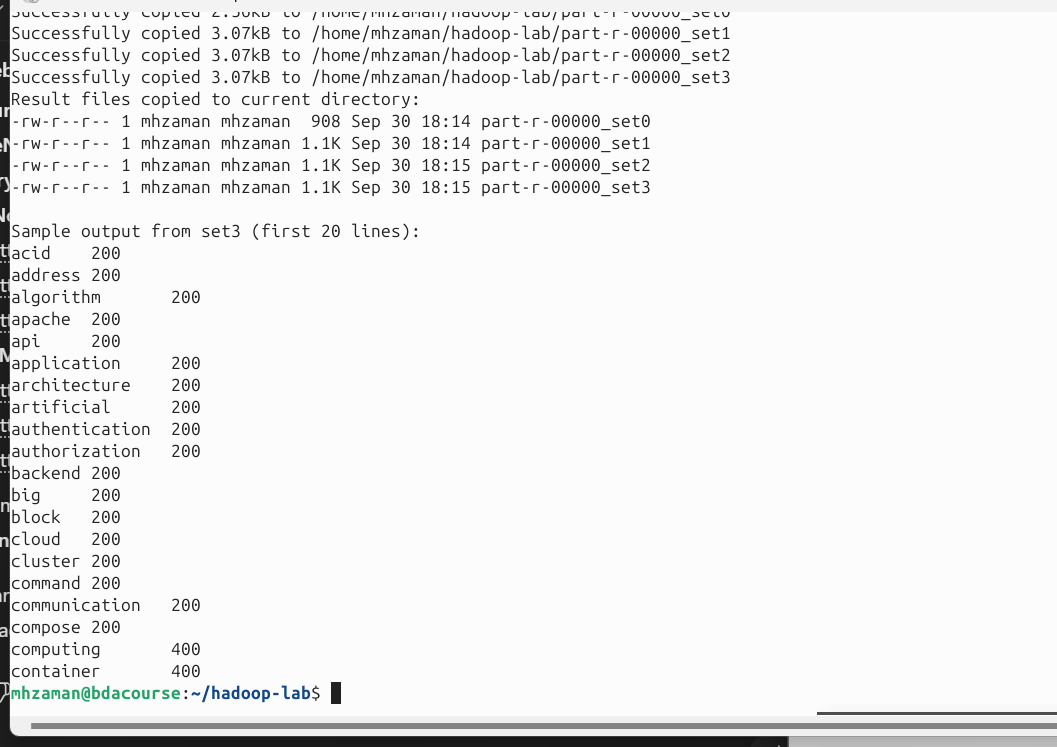
A screen shot of a computer program

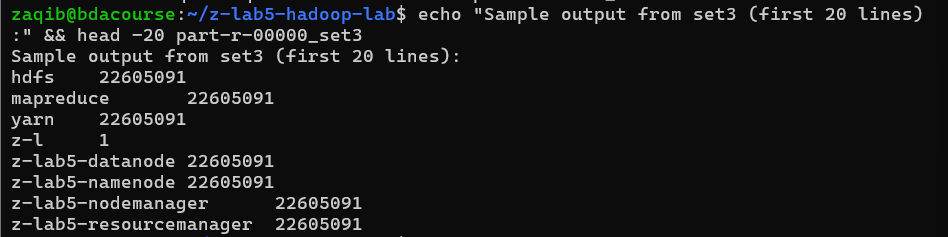
AI-generated content may be incorrect.

SO SEE HOW IT SAYS YES “Z-LAB5-NAMENODE …” SO IT IS ONLY GENERATING TOKENS FROM THOSE IN THE SPEECH MARKS

THAT’S WHY WHEN WE DO HEAD -20 THEN IT ONLY HOWS THESE, ALL WITH SAME COUNTS AS IT ONLY GENERATED FROM THOSE TOKENS





**SCREENSHOT 6:** Capture the sample output showing word counts.

### STEP 11: Check HDFS Health and Metrics

**What this does:**

* **hdfs dfs -du**: Shows disk usage per directory in HDFS
* **hdfs dfs -count**: Displays file count, directory count, and total size
* **hdfs dfsadmin -report**: Shows DataNode status, capacity, and usage
* **hdfs fsck**: Performs filesystem check showing blocks and their locations

docker exec namenode bash -c '

echo "═══════════════════════════════════════════"

echo "HDFS DISK USAGE"

echo "═══════════════════════════════════════════"

hdfs dfs -du -h /user/inputdata

echo ""

echo "═══════════════════════════════════════════"

echo "HDFS FILE COUNT AND SIZE"

echo "═══════════════════════════════════════════"

hdfs dfs -count -h /user/inputdata

echo ""

echo "═══════════════════════════════════════════"

echo "HDFS CLUSTER REPORT"

echo "═══════════════════════════════════════════"

hdfs dfsadmin -report

echo ""

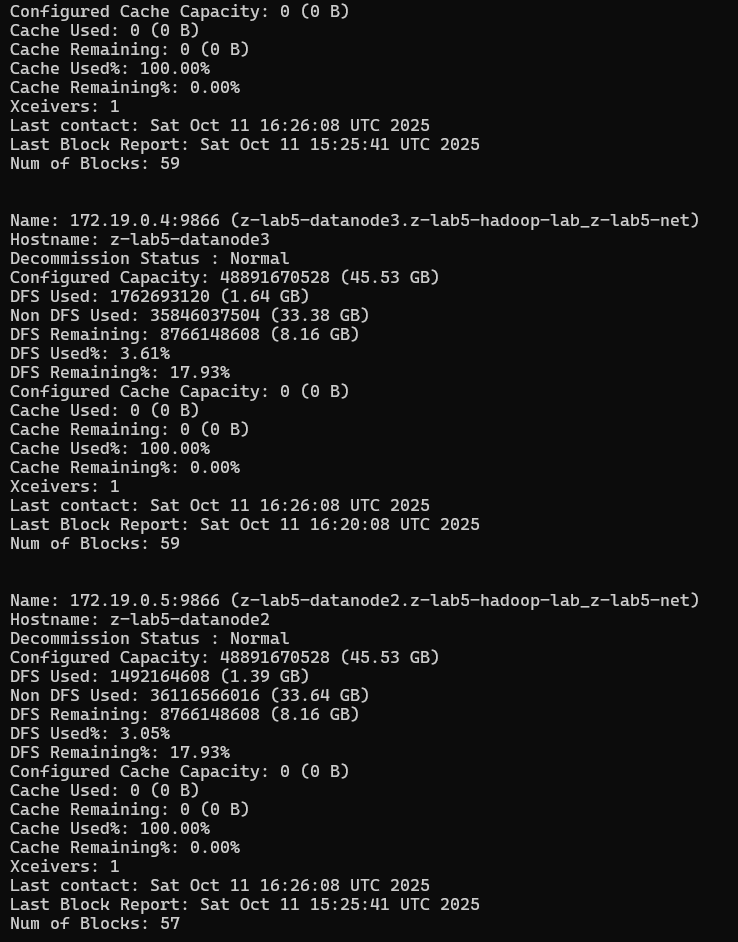
echo "═══════════════════════════════════════════"

echo "HDFS FILESYSTEM CHECK"

echo "═══════════════════════════════════════════"

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

'

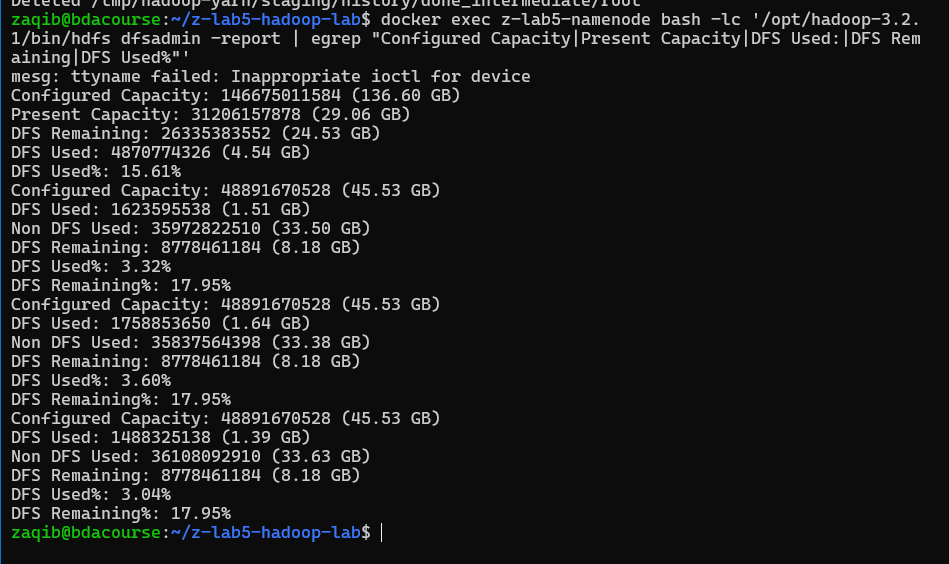
A computer screen with white text

AI-generated content may be incorrect.A screen shot of a computer code

AI-generated content may be incorrect.A computer screen with white text

AI-generated content may be incorrect.

**SCREENSHOT 7:** Capture the HDFS cluster report showing configured capacity and DFS usage.



### STEP 12: Check YARN Applications

**What this does:** Lists all YARN applications (your WordCount jobs) with their IDs, states, and resource usage.

docker exec namenode bash -c '

echo "═══════════════════════════════════════════"

echo "YARN APPLICATIONS LIST"

echo "═══════════════════════════════════════════"

yarn application -list -appStates ALL

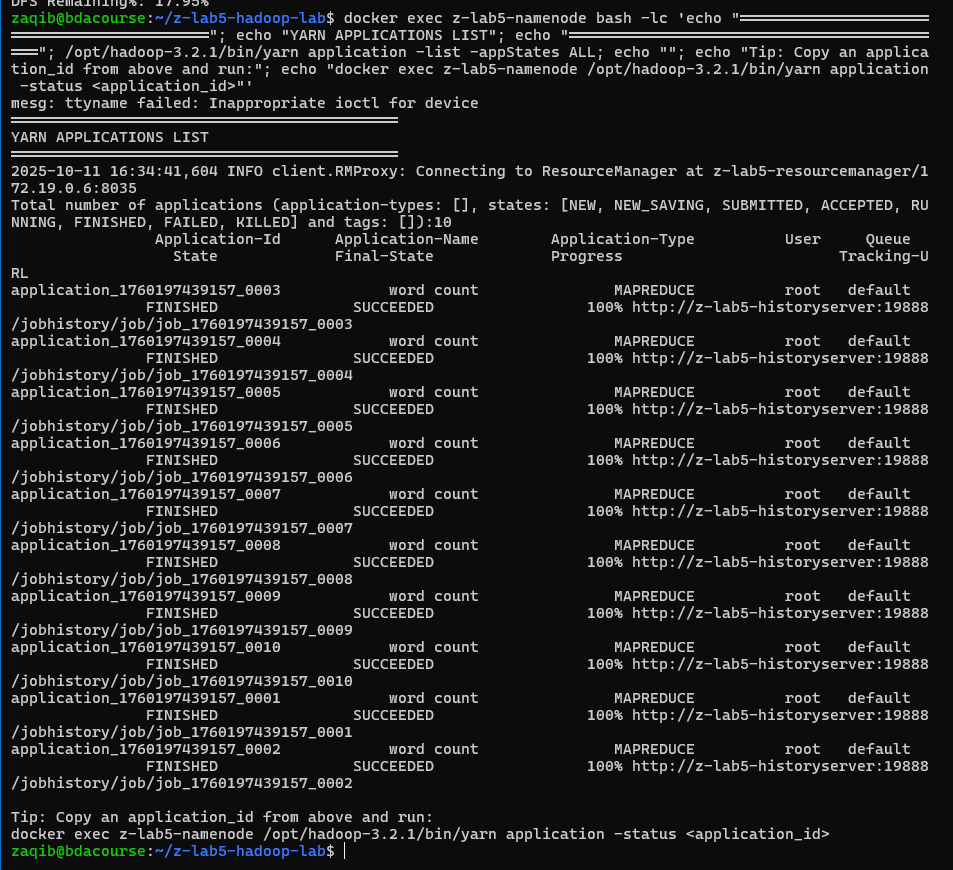
echo ""

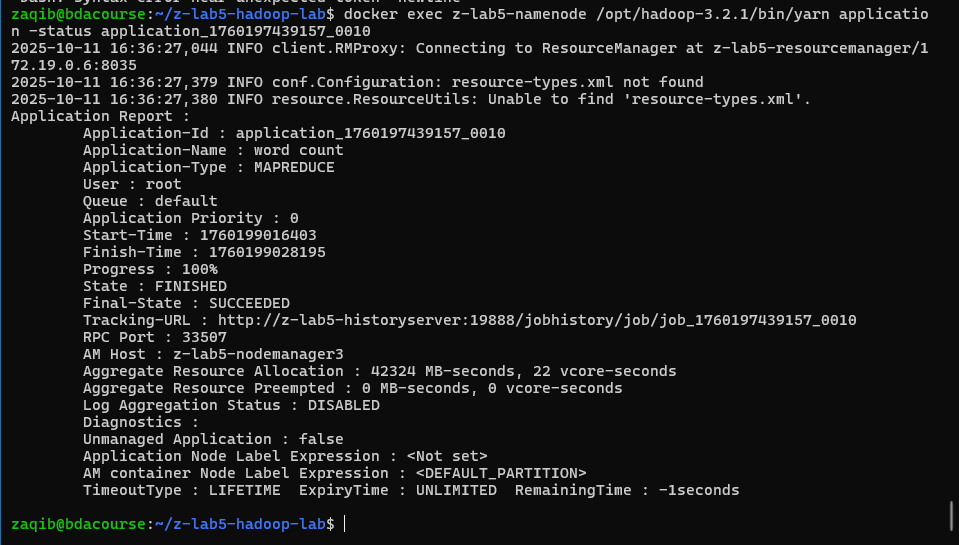
echo "Tip: Copy an application\_id from above and run:"

echo "docker exec namenode yarn application -status <application\_id>"

'

**SCREENSHOT 8:** Paste it





### STEP 13: Access Hadoop Web UI

Open these URLs in your browser (use localhost since you're on Ubuntu):



NameNode UI: http://localhost:9870

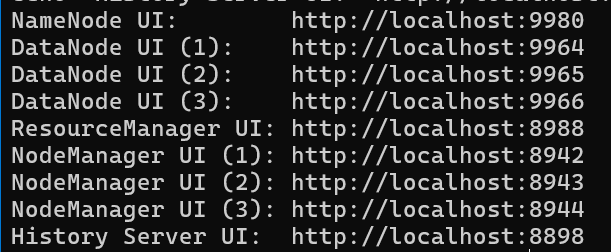
DataNode UI: http://localhost:9864

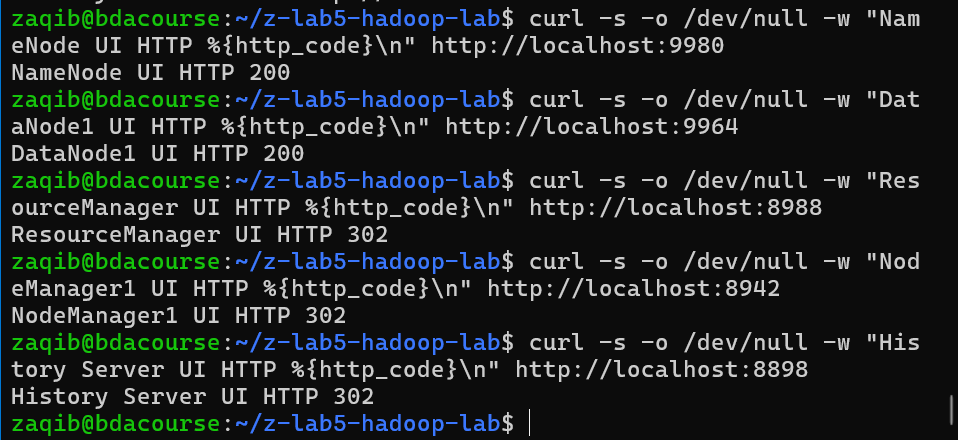
ResourceManager UI: http://localhost:8088

NodeManager UI: http://localhost:8042

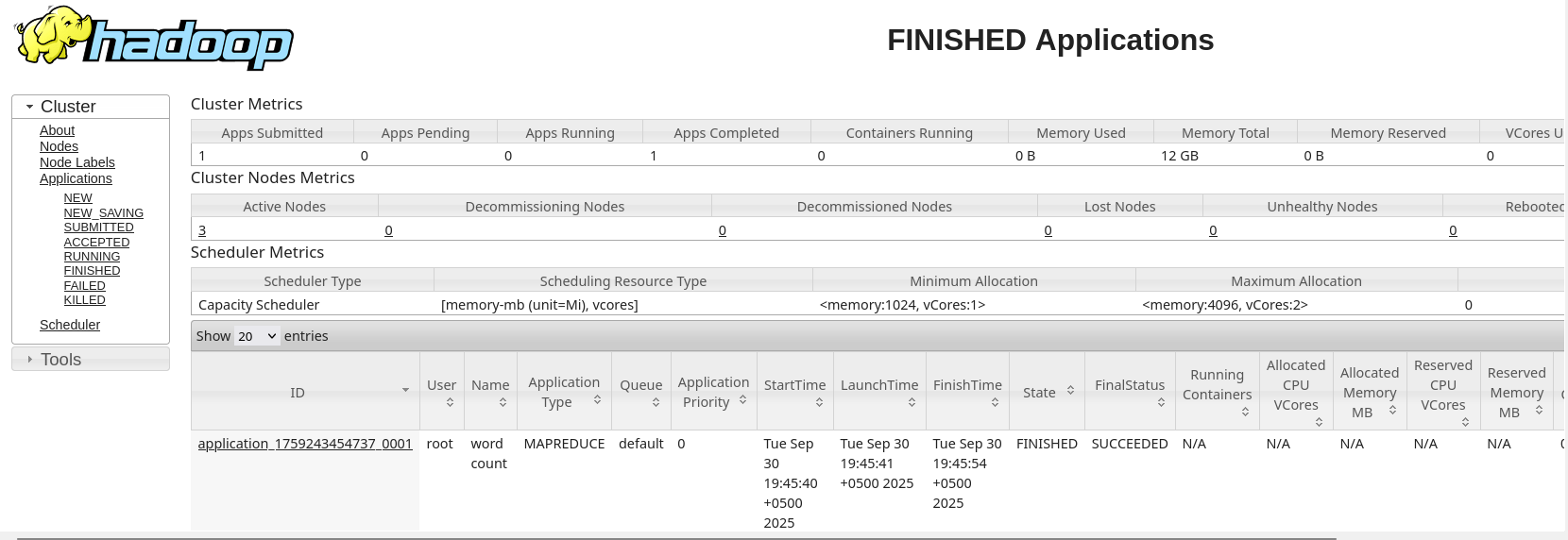
History Server UI: <http://localhost:8188>

FOR ME



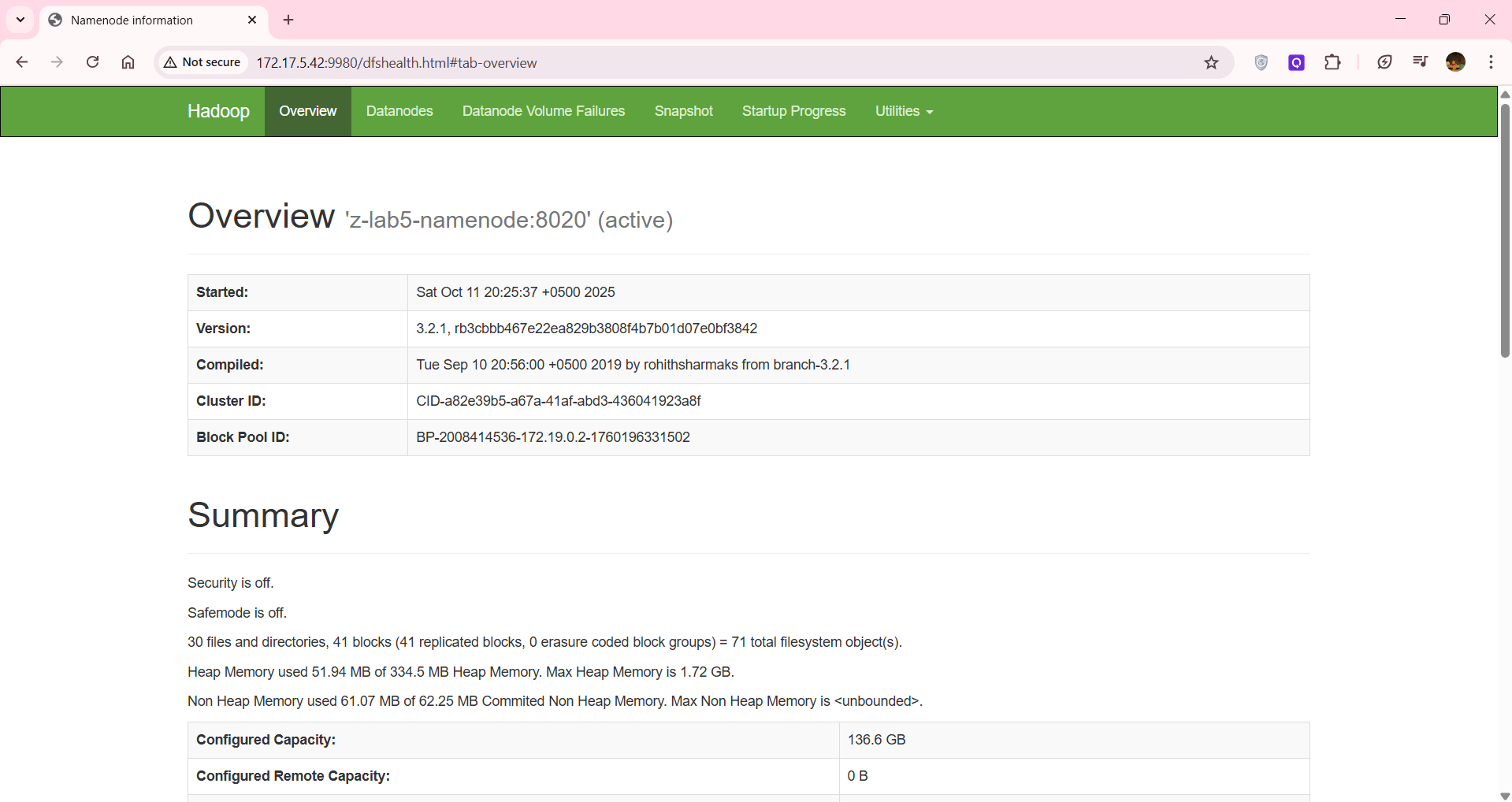


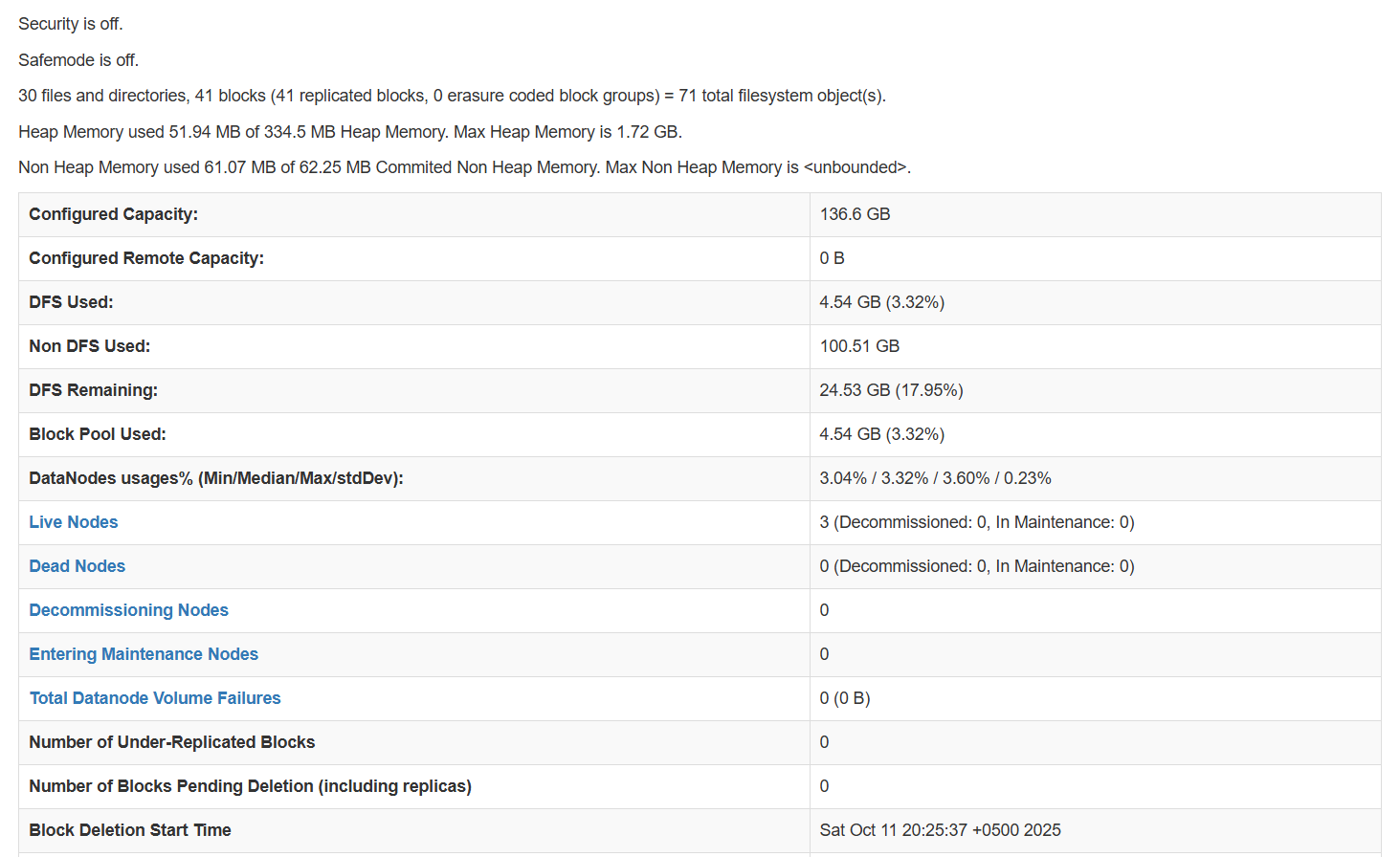
ALL EXIST AND ARE LIVE



**What to capture in each UI:**

1. **NameNode UI (localhost:9870)**
   * Overview page showing Configured Capacity, DFS Usage, Live DataNodes



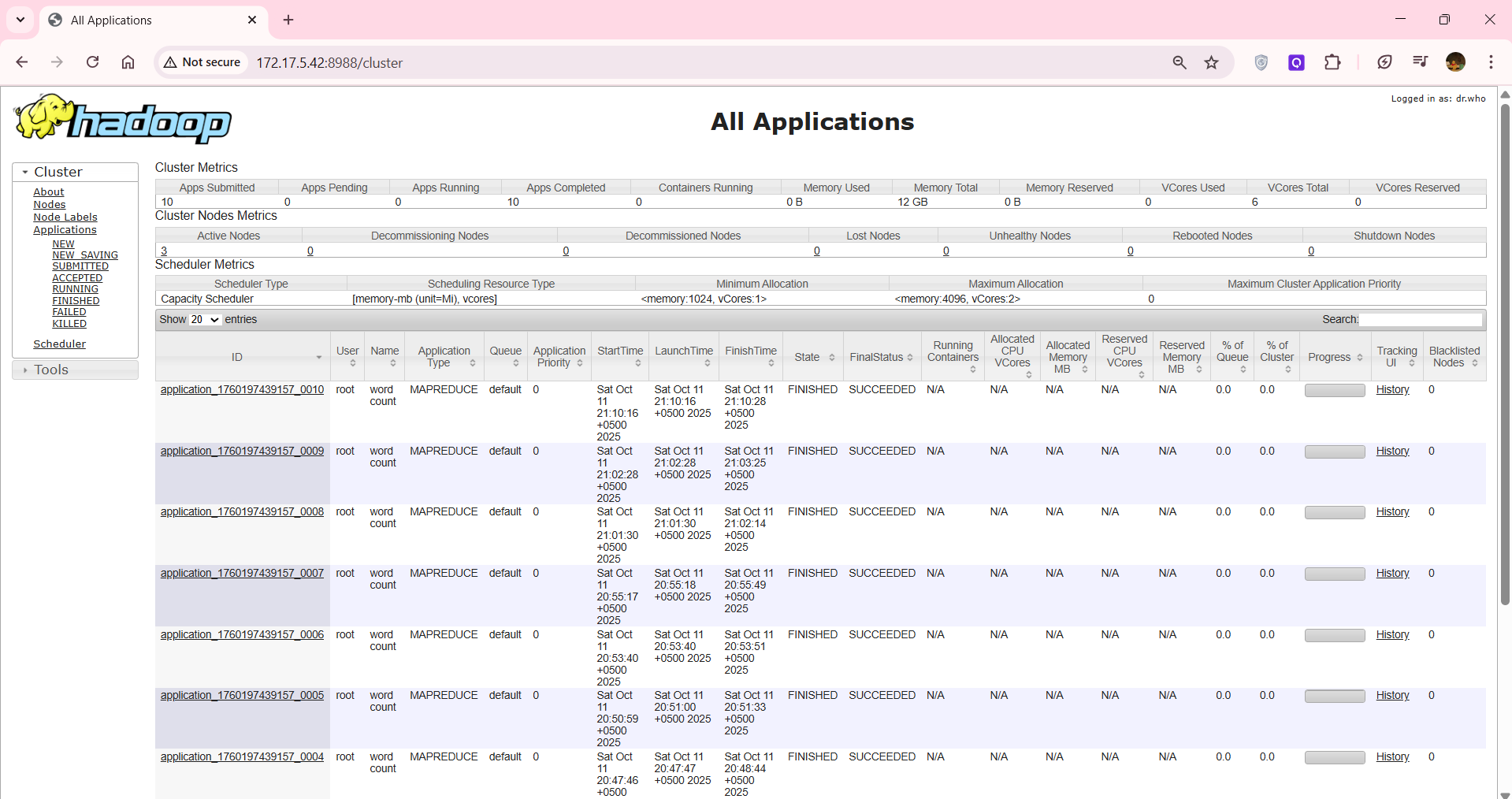


* + Navigate to Datanodes tab to see all 3 datanodes

A screenshot of a computer

AI-generated content may be incorrect.

1. **ResourceManager UI (localhost:8088)**



* + Click on "Applications" to see your 4 WordCount jobs

A screenshot of a computer

AI-generated content may be incorrect.

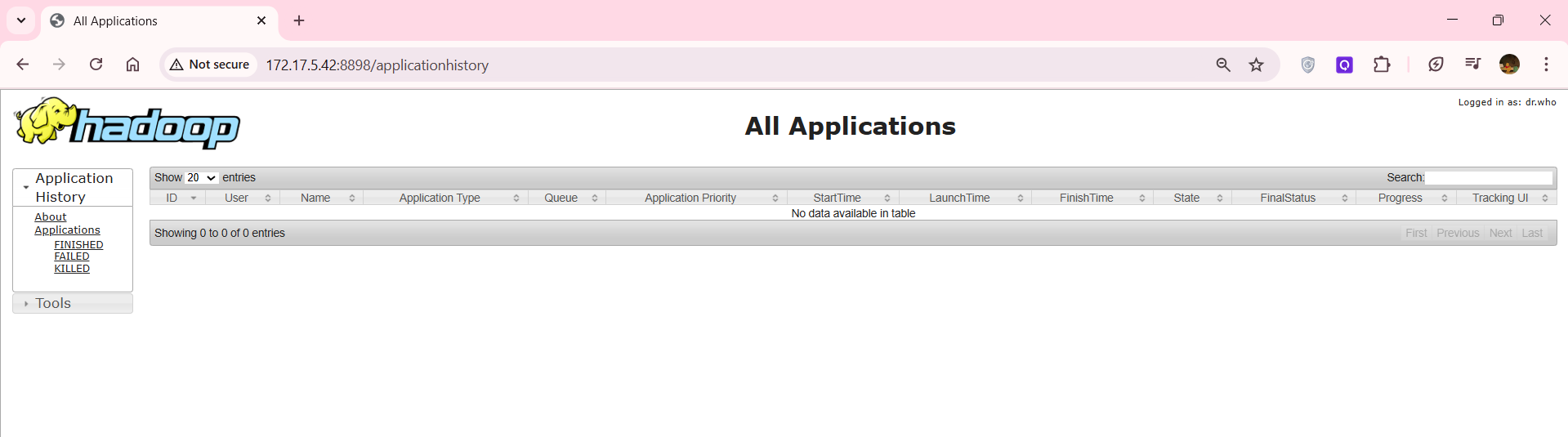
ITS MORE BECAUSE I DID IT BAAR BAAR TO CHECK IF IT WORKING…

* + Click on any job to see detailed metrics

A screenshot of a computer

AI-generated content may be incorrect.

1. **History Server UI (localhost:8188)**



* + Shows completed job history with detailed counters and timing

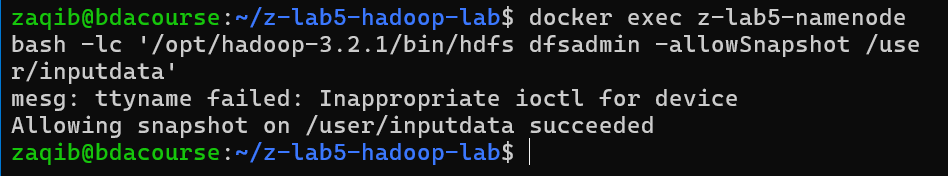
**SCREENSHOT 9-13:** Capture each of the 5 web UIs showing relevant metrics. Explain a bit.

The Hadoop UIs confirm the whole lab worked: the NameNode overview shows HDFS is healthy and writable, with total configured capacity, clear DFS Used and Remaining values, and three live DataNodes. The Datanodes page lists each node as healthy with recent “last contact,” their capacities, blocks stored, and small, balanced usage. The ResourceManager “All Applications” page lists your WordCount jobs as FINISHED with SUCCEEDED status, including start, finish, and links to “History.” Opening a single application shows its ID, user, queue, elapsed runtime, and the NodeManager that hosted the ApplicationMaster. The History Server provides the permanent record of completed jobs and exposes detailed counters and timelines. Together, these screens prove the cluster is up, storage is available, all nodes are connected, and your MapReduce jobs ran to completion successfully.

### STEP 14: Clean HDFS Snapshots and Advanced Commands

**What this does:**

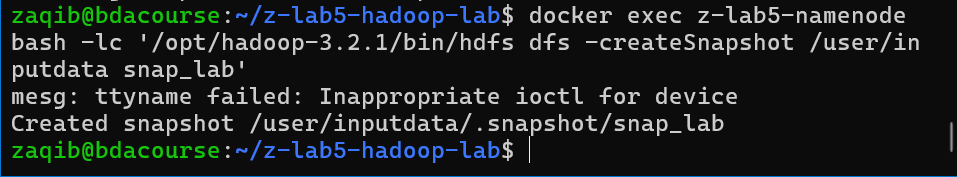
* **createSnapshot**: Creates a point-in-time snapshot of the directory (for backup/recovery)
* **setrep**: Sets the replication factor (number of copies) for a file
* **fsck**: Checks filesystem health and block replication
* **deleteSnapshot**: Removes the snapshot



docker exec namenode bash -c '

echo "Creating HDFS snapshot..."

hdfs dfs -createSnapshot /user/inputdata snap\_lab



echo "Listing snapshots:"

hdfs dfs -ls /user/inputdata/.snapshot

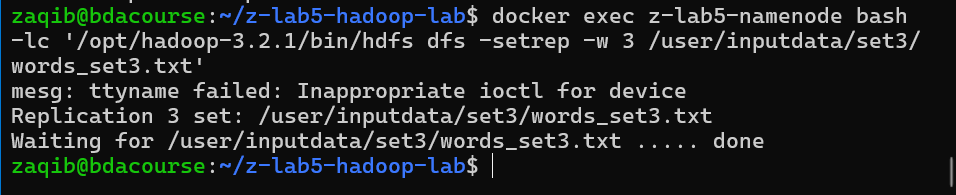
A screen shot of a computer

AI-generated content may be incorrect.

echo ""

echo "Setting replication factor for set3:"

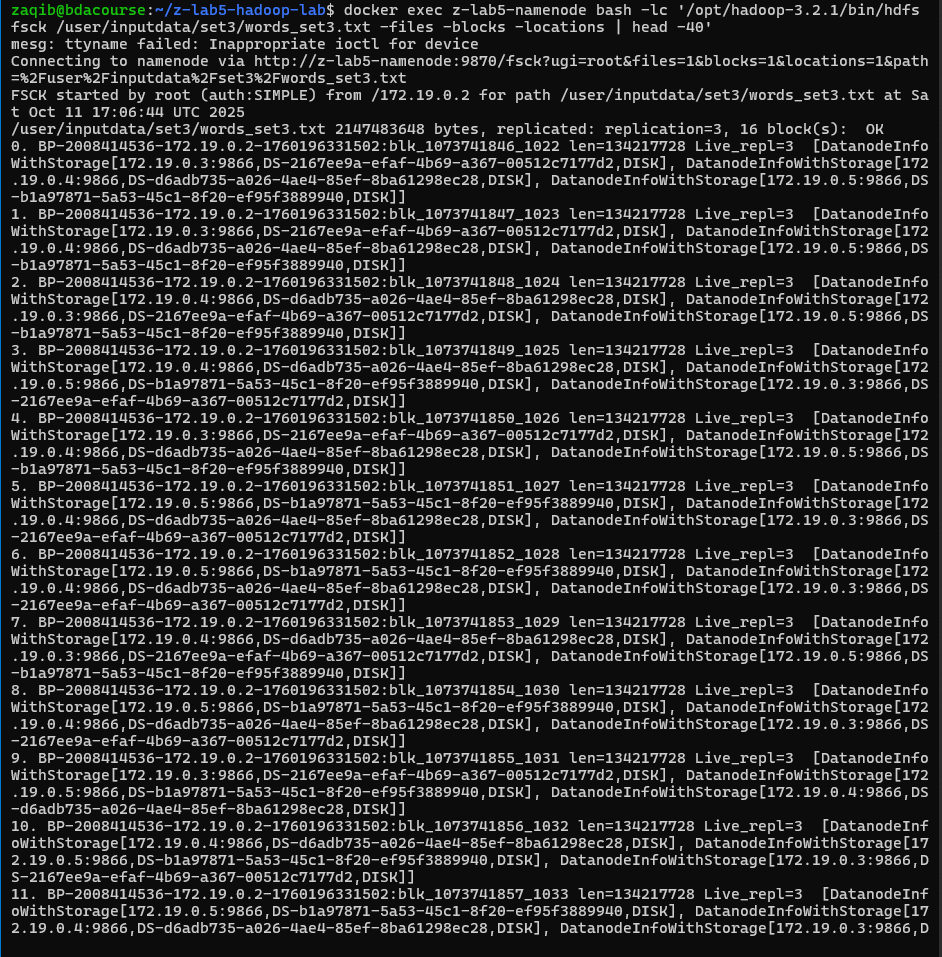
hdfs dfs -setrep -w 3 /user/inputdata/set3/words\_set3.txt



echo ""

echo "Checking block replication:"

hdfs fsck /user/inputdata/set3/words\_set3.txt -files -blocks

A screenshot of a computer program

AI-generated content may be incorrect.

echo ""

echo "Deleting snapshot:"

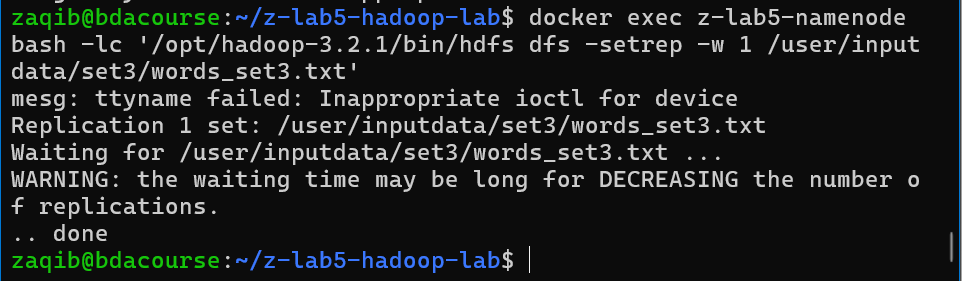
hdfs dfs -deleteSnapshot /user/inputdata snap\_lab

A screenshot of a computer code

AI-generated content may be incorrect.

echo "Advanced HDFS operations completed"

'



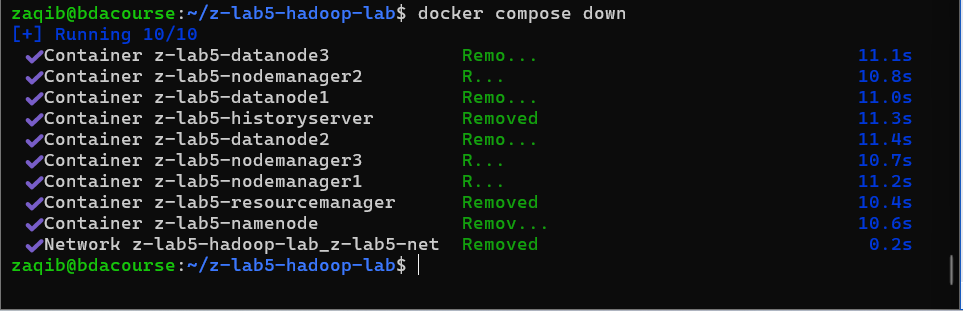
**SCREENSHOT 14:** Capture the snapshot operations and try to explain a bit

### STEP 16: Stop and Cleanup (After completing the lab)

**What this does:** Gracefully stops all Hadoop services. Data is preserved unless you uncomment the rm command.

# Stop all containers

docker compose down



# Optional: Remove all data (if you want to completely reset)

# sudo rm -rf ./data

echo "Hadoop cluster stopped"

echo "Data directories preserved in ./data for restart"

### Quick Reference Commands

# Restart the cluster

docker compose up -d

# Check container status

docker ps

# View container logs

docker logs namenode

docker logs resourcemanager

# Execute commands inside namenode

docker exec -it namenode bash

# Inside namenode, useful HDFS commands:

hdfs dfs -ls /

hdfs dfs -du -h /user

hdfs dfsadmin -report

yarn application -list

# Exit namenode shell

exit

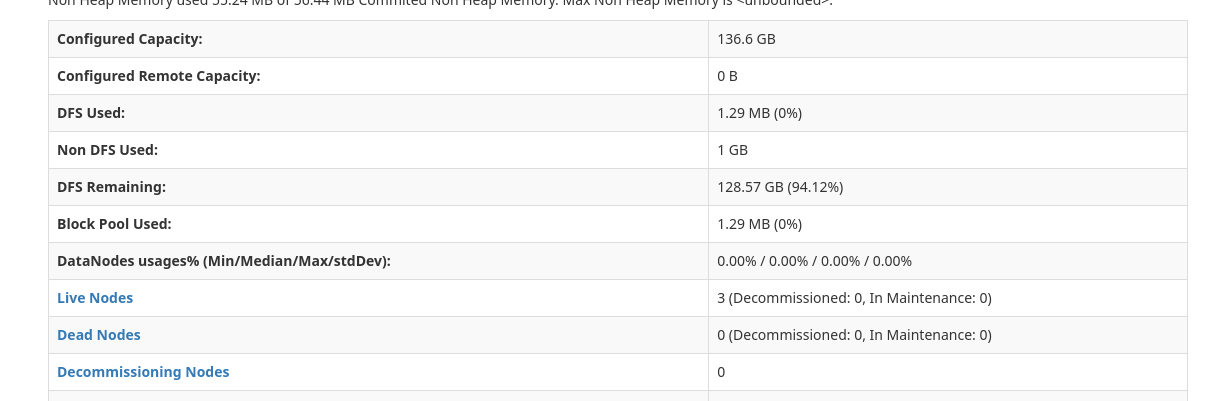
**Lab Report Checklist**

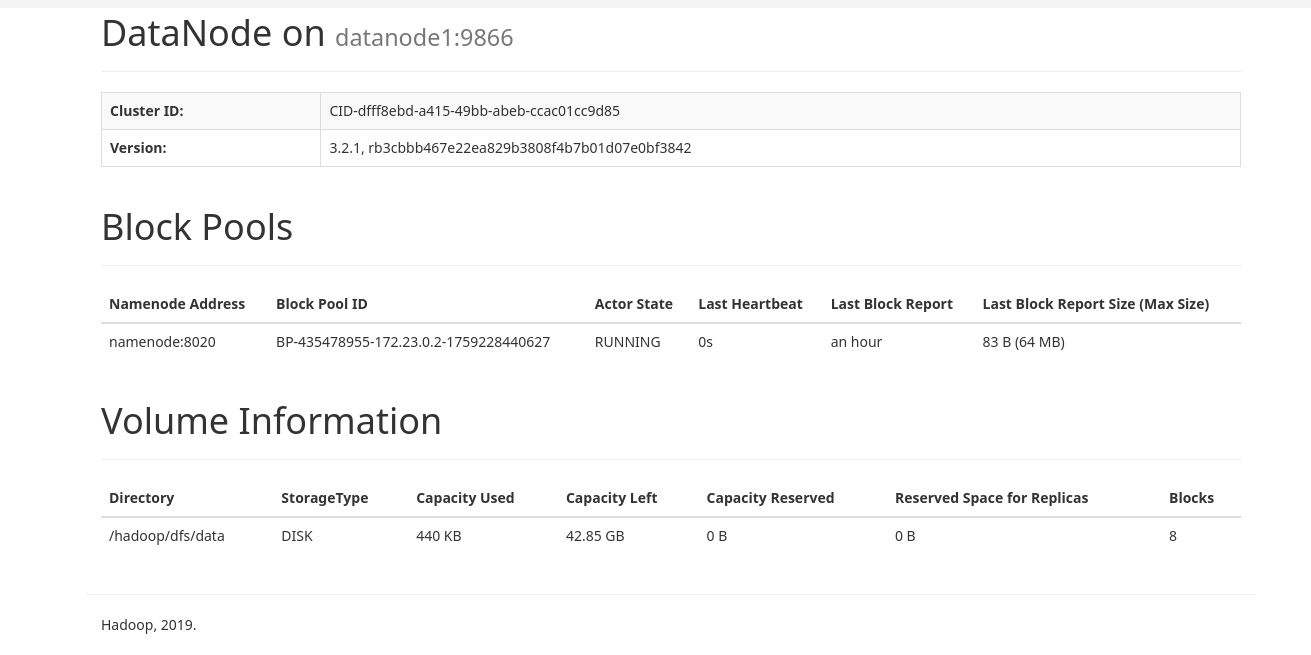
Screenshots of:

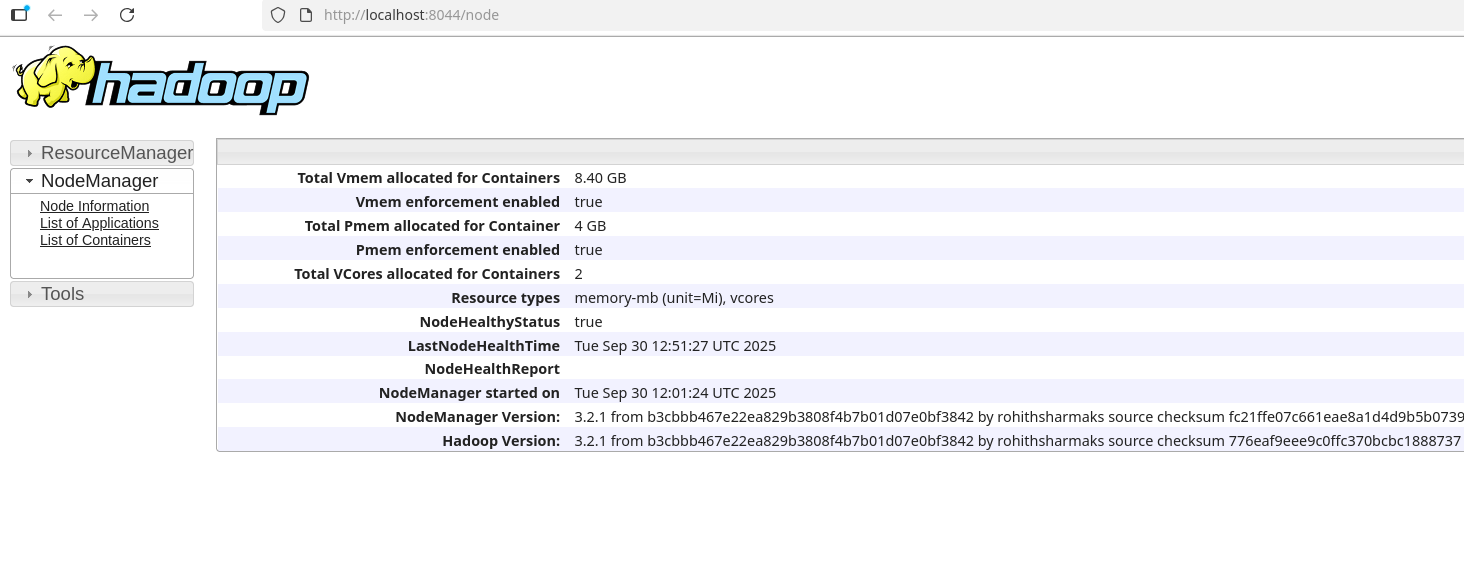
1. HDFS directory listing with uploaded files
2. WordCount job completion for set0
3. WordCount job completion for set1
4. WordCount job completion for set2
5. WordCount job completion for set3
6. Sample word count output
7. HDFS cluster report
8. YARN applications list
9. NameNode Web UI
10. ResourceManager Web UI
11. History Server Web UI
12. DataNode metrics
13. NodeManager metrics
14. HDFS snapshot operation

**To access the UI:**

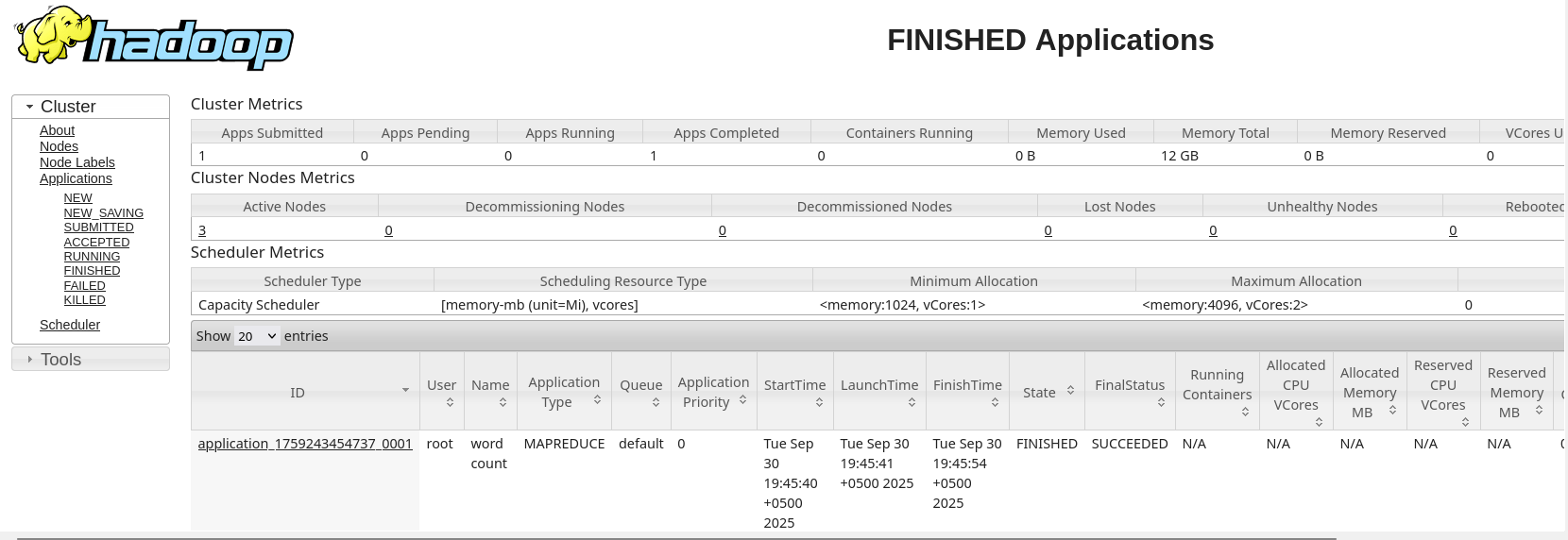
* **for s in "NameNode:9870" "ResourceManager:8088" "HistoryServer:8188" "DataNode1:9864" "DataNode2:9865" "DataNode3:9866" "NodeManager1:8042" "NodeManager2:8043" "NodeManager3:8044"; do svc=${s%%:\*}; port=${s##\*:}; url="http://localhost:$port"; status=$(curl -s -o /dev/null -w "%{http\_code}" $url); if [ "$status" = "200" ] || [ "$status" = "302" ]; then echo "$svc → $url → UP ($status)"; else echo "$svc → $url → DOWN ($status)"; fi; done**

****

****

****

**Final Checkpoint:**

****